UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Blaž Urankar

Spletna aplikacija za osebnostno rast

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2016

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Blaž Urankar

Spletna aplikacija za osebnostno rast

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Aleš Smrdel

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Sodobni čas daje vedno več poudarka na človekovo fizično plat, zanemarja pa njegovo psihično plat. Ker je človek psihofizično bitje, je za njegovo dobro počutje potrebno poskrbeti za obe plati. V diplomskem delu razvijte aplikacijo, ki bo uporabnikom pomagala krepiti njihovo samozavest ter jim pomagala pri njihovi osebnostni rasti. Pri razvoju aplikacije na strani odjemalca uporabite tehnologije za oblikovanje strani v kombinaciji s tehnologijami, ki omogočajo izdelavo odzivnih in bogatih spletnih aplikacij. Na strani strežnika pa pri razvoju uporabite tehnologijo Node.js. Pri uporabi bo razvita aplikacija potrebovala tudi podatke, ki naj jih pridobi iz baze podatkov. V ta namen uporabite neko tehnologijo, ki omogočajo upravljanje s podatkovnimi bazami. Za to, da bo aplikacija funkcionalna pa poskrbite tudi za ustrezne vsebine. Razvito aplikacijo pa ponudite v testiranje tudi uporabnikom in zabeležite ter opišite njihove odzive na aplikacijo.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Blaž Urankar sem avtor diplomskega dela z naslovom:

Spletna aplikacija za osebnostno rast

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Aleša Smrdela,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 4. marca 2016 Podpis avtorja:

Zahvaljujem se mentorju, ker si je vedno vzel čas zame, mi hitro odgovarjal na vprašanja in mi odlično svetoval. Zahvaljujem se tudi mojim prijateljem, ki mi vsak dan znova pokažejo zabavno plat življenja. Brez njih ne bi prilezel tako daleč. Najbolj se moram zahvaliti svojima staršema. Ko v študiju nisem videl smisla, sta mi ga uspela pokazati. Ta diploma je za vaju.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
JWT	JavaScript Web Token	Spletni žeton JavaScript
HTML	HyperText Markup Language	Jezik za označevanje nadbesedila
CSS	Cascading Style Sheets	Kaskadne slogovne pole
SQL	Structured Query Language	Strukturiran jezik za poizvedovanje
NoSQL	Non-relational Structured Query Language	Strukturiran jezik za poizvedovanje brez relacij
JSON	JavaScript Object Notation	Zapis objektov JavaScript
BSON	Binary JavaScript Object Notation	Binarni zapis objektov JavaScript
NPM	Node Package Manager	Upravljalec paketov Node
MEAN	MongoDB, Express.js, AngularJS, Nodejs	Kombinacija tehnologij MongoDB, Express.js, AngularJS, Nodejs

Povzetek

Naslov: Spletna aplikacija za osebnostno rast.

V diplomski nalogi je predstavljen postopek zasnove in izdelave spletne aplikacije za osebnostno rast. Živimo v dobi, ko je za naše fizično udobje dobro poskrbljeno, v nevarnosti pa je naš um. Ljudje se pogosto počutijo manjvredne in ne razumejo svojih čustev, zato smo razvili spletno aplikacijo, ki pomaga krepiti samozavest in vzpodbuja razumevanje čustev. Razvijanja smo se lotili z uporabo nekaterih novejših tehnologij. Na strani strežnika smo uporabili tehnologije Node.js, Express.js, in MongoDB, na strani odjemalca pa AngularJS, HTML, CSS in JavaScript. Skozi leta smo nabrali veliko idej s psiholoških področij, ki v človeku vzbudijo pozitivna razkritja. Te ideje smo vključili v aplikacijo in jih predstavili v pričujočem delu. Predstavljajo osnovo za pozitivno okolje, ki raste skupaj s svojimi uporabniki. V diplomo smo dodali tudi sklepne ugotovitve, mnenja testnih uporabnikov in cilje za nadaljnje delo.

Ključne besede: spletna aplikacija, psihologija, osebnostna rast, čustva, samozavest.

Abstract

Title: Web application for personal growth.

The thesis presents the process of designing and producing a web application for personal growth. We live in an era where it is well taken care of our physical comfort, but our mind is in danger. People often feel worthless and do not understand their feelings so we have developed a web application that helps strenghten confidence and encourages understanding of emotions. We started development with the use of some newer technologies. On the server side, we used Node.js, Express.js and MongoDB technologies while we used AngularJS, HTML, CSS and JavaScript on the client side. Over the years we have accumulated a lot of ideas from psychological fields that inspire positive revelations. We included these ideas in the application and we also presented them in this work. They provide the basis for a positive environment that grows together with it's users. We also added conclusions, opinions of test users and objectives for future work into the thesis.

Keywords: web application, psychology, personal growth, feelings, self-esteem.

Poglavje 1 Uvod

Živimo v dobi, ko je za naše fizično udobje dobro poskrbljeno. Morda celo preveč dobro. Glavni vzroki smrti so večinoma povezani z boleznimi, ki nastanejo zaradi pomanjkanja telesne aktivnosti, prekomerne teže in uživanja različnih substanc. Smrtonosnih plenilcev v razvitem svetu ni več, zato ni potrebe, da bi imeli telo pripravljeno za tek ali borbo za preživetje in redko kdo ga tudi ima, pa čeprav imamo na voljo več načinov kot kadarkoli prej, da pridobimo izklesano telo. Obiščemo lahko celo plastičnega kirurga, ki nam dele obraza spremeni natanko tako, kot si želimo. Dozdeva se nam, da nevarnosti za nas ni, a realnost je drugačna. Pozabljamo na nekaj veliko bolj pomembnega, naš um. Možgani so del našega telesa in delujejo kot računalnik, um pa je program, ki predstavlja to, kar smo. Če možgane prizadene bolezen in ne delujejo več normalno, to vpliva na delovanje našega uma, vendar ga ne spremeni. Program je še vedno napisan enako. To velja tudi za psihološke motnje, kot sta bipolarnost in anksioza. Njune simptome lahko zdravimo s tabletami, a um se zaradi tega ne bo spremenil, zato ga je potrebno zdraviti drugače. Potrebno je spremeniti program, ki ga izvajajo naši možgani.

Tu nastane problem, saj ljudje pogosto ne razumejo kaj čutijo, zakaj to čutijo in kako si lahko pomagajo, čeprav je to razloženo v literaturi[1]. Premik iz osebnih na materialne vrednote je povzročil, da se ljudje počutijo prazne, brez vrednosti in posledično brez prave samozavesti. Za ekonomijo je to blagoslov, saj se s ponujanjem materialnih vrednot odlično zasluži, tu pa verjetno malokdo gleda na to, kaj je pravzaprav zdravo za človeka.

Odločili smo se, da glede tega nekaj storimo. V mladosti so nas, tako kot ogromno ljudi, ovirala različna mišljenja, strahovi in dvomi, s katerimi pa se nismo želeli sprijazniti, zato smo začeli s potjo čustvenega in osebnostnega razvoja. Na podlagi združevanja računalniških tehnologij in psiholoških dognanj, nabranih v zadnjih letih, smo se odločili narediti aplikacijo, katere namen je človeku okrepiti samozavest, ga spoznati z njegovimi čustvi in mu omogočiti osebnostni razvoj. To storimo s tako imenovanimi rezinami, ki vsebujejo vse potrebno, da človeku razložijo neko idejo, ga pozneje nanjo spominjajo ter mu podajo napotke, kako jo uresničevati v praksi. Na spletu je prisotnih vse več ljudi vse več časa, kar nas je vzpodbudilo, da naredimo aplikacijo široko dostopno.

Z namenom, da ljudem približamo aplikacijo, smo se jo odločili razviti kot spletno aplikacijo. Pri izdelavi aplikacije smo na strežniku uporabili tehnologije Node.js, kot izvajalno okolje, ogrodje Express.js, kot vmesnik med odjemalcem in podatkovno bazo MongoDB ter module Mongoose.js, Faker, Passport.js in JWT. Na strani odjemalca smo uporabili tehnologije Git za nadzor verzij izvorne kode in še JavaScript, jQuery, CSS in HTML, ki smo ga nadgradili z uporabo ogrodja AngularJS, ki uporabniku ponuja izredno hitro odzivnost aplikacije in funkcionalnosti, ki drugače niso možne.

Ker je psihologija zelo širok koncept in ponuja različne alternative, poleg tega pa je človekova notranjost nekaj, kar se ne da »popredalčkati«, smo dodali možnost, da uporabniki sami dodajajo rezine, ki so za tem na voljo tudi drugim uporabnikom. S tem želimo ustvariti okolje za osebnostno rast, ki raste skupaj s svojimi uporabniki.

Poglavje 2 Uporabljena orodja in tehnologije

Med študijem smo se naučili osnove jezikov HTML in CSS ter nekaj malega o funkcionalnosti skriptnega jezika JavaScript. Zanima nas razvoj spletnih aplikacij, zato smo se odločili, da poglobimo obstoječa znanja ter uporabimo še kaj bolj konkretnega.

Za delovanje aplikacije na strani strežnika smo preizkusili več tehnologij, a smo se na koncu odločili za izvajalno okolje Node.js v kombinaciji z ogrodjem Express.js. Ključna prednost je ta, da se uporablja izključno JavaScript, kar omogoča, da aplikacijo razvijamo z le enim programskim jezikom. To se nam je, kot začetnikom, zdelo izredno pomembno, saj smo osnove JavaScripta že poznali. Tako smo se lahko bolj osredotočili na ustvarjalni vidik razvoja in seveda na pisanje psihološke vsebine.

Imeli smo že kar nekaj izkušenj z uporabo tehnologije MySQL, ki smo jih pridobili med študijem, in pisanje stavkov SQL nam ni bilo tuje, a vendarle se tokrat nismo odločili za tak pristop. Uporabili smo podatkovno bazo MongoDB, ki uporablja mehanizem NoSQL, za kar obstajajo trije razlogi. Prvi je ta, da se na internetu pogosto omenja izraz MEAN, ki v osnovi predstavlja razvijanje z uporabo tehnologije MongoDB in še treh drugih, Express.js, AngularJS in Node.js, za katere smo se odločili že pred tem. To vsekakor ni bil najbolj pomemben razlog, saj MongoDB brez problema deluje tudi v kombinaciji z drugimi tehnologijami. Drugi razlog je ta, da je mehanizem NoSQL v zadnjih letih s prihodom novih tehnologij postal izredno popularen in se ga, kot razvijalcu nove generacije, vsekakor splača imeti v repertoarju. Najbolj pomemben razlog pa je način, kako NoSQL deluje. V dokumente podatkovne baze lahko brez težav shranimo kakršnekoli podatke, prav tako ni potrebno vnaprej definirati sheme podatkovne baze, vendar lahko kadarkoli dodamo poljubno zbirko s poljubnimi podatki. Aplikacija Čutilko je bil naš prvi samostojni obsežni projekt, zato smo pričakovali ogromno sprememb pri načrtovanju podatkovne baze, tu pa se je resnično pokazala vrednost podatkovne baze MongoDB. Bazo smo lahko gradili sproti, ko smo šele odkrivali nove elemente aplikacije.

Za povezavo med MongoDB in Express.js smo uporabili modul Mongoose.js, s katerim lahko definiramo sheme dokumentov v podatkovni bazi, na voljo pa ima tudi ogromno koristnih metod za ustvarjanje in upravljanje s podatki.

Za delovanje aplikacije na strani odjemalca smo izbrali tehnologijo AngularJS, ki nadgradi kodo HTMLja in omogoča izredno odzivnost aplikacije. Mnenja smo, da ima za začetnika zelo zahtevno krivuljo učenja, a so rezultati vredni truda. Želeli smo uporabiti tudi kakšno popularno ogrodje za oblikovanje strani, vendar bi bila s tem naša aplikacija preveč podobna ostalim, zato smo se malo bolj poglobili v tehnologijo CSS in aplikacijo oblikovali po svoje.

Za nadzor različic programske kode smo uporabili sistem Git, ki omogoča, da v katerem koli trenutku shranimo trenutno stanje kode in se po potrebi pozneje v to stanje vrnemo. S tem smo prihranili ogromno časa, ko smo del kode nezaželeno spremenili ali pa z njo nismo bili zadovoljni.

2.1 Node.js, NPM in Express.js

Node.js[2] je izvajalno okolje, ki ga je leta 2009 razvil Ryan Dahl. Zgrajeno je na odprto-kodnem gonilniku Chrome V8[3]. Njegova glavna lastnost je zmožnost interpretacije in izvršitve kode skriptnega jezika JavaScript, poleg tega je izredno hiter in učinkovit ter deluje na vseh najbolj popularnih operacijskih sistem. Odličen je za uporabo v aplikacijah, ki podatke izmenjujejo v realnem času. Zmožen je visoke prepustnosti ogromnega števila dvosmernih povezav, kjer lahko s komunikacijo začne tako odjemalec kot strežnik. Deluje asinhrono, kar pomeni, da ob zahtevi ne blokira delovanje programa, temveč se posveti drugim opravilom, ko pa je zahteva zaključena, izvrši njeno izhodno metodo[4]. Leta 2011 je bil razvit tudi upravljalec paketov Node (angl. NPM)[5], ki se namesti sočasno z Node.js. Omogoča namestitev katerekoli JavaScript knjižnice, ki jo najdemo v njegovi ogromni zbirki. Vsebuje praktično vse, kar bi potrebovali za začetek kateregakoli projekta. Express.js[6] je najbolj popularno minimalistično ogrodje za Node.js, namenjeno izjemno hitremu razvoju strežniškega dela spletnih aplikacij, z ogromno koristnimi metodami in odlično podporo na spletu. Tako kot Node.js je napisan izključno v programskem jeziku JavaScript in deluje na vseh najbolj popularnih operacijskih sistemih. Z njim lahko ustvarimo enostranske, večstranske in hibridne mobilne aplikacije, izvrsten pa je tudi pri delovanju kot vmesnik med podatkovno bazo in odjemalnim delom aplikacije.

2.2 MongoDB in Mongoose.js

MongoDB[7] je podatkovna baza, ki uporablja mehanizem NoSQL. Od relacijskih podatkovnih baz se med drugim razlikuje po tem, da namesto tabel in vrstic uporablja zbirke in dokumente. Zbirke vsebujejo množice dokumentov, ki so sestavljeni iz kombinacij ključev in vrednosti, natanko tako, kot pri JSON formatu. Sheme podatkovne baze se lahko prosto

spreminjajo in jih ni potrebno definirati. Podatke lahko namreč vstavljamo čisto poljubno, brez predhodnega definiranja in brez omejitev. Baza podatke shranjuje v formatu BSON, ki je le binarni prikaz formata JSON. MongoDB je leta 2009 postal na voljo kot odprtokodni projekt, ki je skozi leta postal izredno popularen[8]. Mongoose.js[9] je modul, ki deluje kot vmesnik med podatkovno bazo in strežniškim delom aplikacije. Omogoča modeliranje podatkov, definiranje shem ter uporabo metod za upravljanje s podatki. Njegova uporaba ni nujna, lahko bi uporabljali osnoven gonilnik podatkovne baze MongoDB, vendar je razvijanje tako veliko lažje, saj ni potrebno vedeti, kaj se dogaja v ozadju.

2.3 Passport in JWT

Passport[10] je vmesni sistem za avtentikacijo uporabnika na strežniku, ki nevpisanim uporabnikom preprečuje izvajanje izbranih zahtev (npr. GET). Danes obstaja več načinov, kako se uporabnik lahko vpiše. Passport uporablja različne strategije, med katerimi se razvijalec lahko odloči, ali jih bo dodal v aplikacijo. Od tipične lokalne strategije, ki zahteva vpis z uporabniškim imenom in geslom, do vpisa z uporabniškimi računi drugih aplikacij, kot so najbolj popularna socialna omrežja. Zaradi novega načina avtentikacije je danes pogosta uporaba žetonov. JWT (JavaScript Web Token)[11] uporablja digitalno podpisane objekte JSON za prenos pomembnih informacij med dvema stranema. Vsebuje vse potrebne podatke o uporabniku in je izredno majhne velikosti, kar omogoča hiter prenos. Žeton oziroma objekt JSON je sestavljen iz glave, ki po navadi vsebuje tip žetona in algoritem kodiranja, tovor, ki vsebuje ključne podatke, na primer ime in ključ avtorja, ter podpis, ki je produkt kodiranja in podpisovanja podatkov iz glave in jedra, združeno s skrivnostjo, ki je po navadi okoljska spremenljivka.

2.4 AngularJS

AngularJS[12] je ogrodje za izdelavo aplikacije na strani odjemalca. Njegova glavna zmožnost je razširitev sintakse HTML, ki omogoča razvoj in uporabo funkcionalnosti, ki drugače niso na voljo. Na voljo ima kar nekaj komponent, ki omogočajo razvoj posebnih funkcionalnosti. Mi smo jih uporabili le nekaj:

- **Storitev** je komponenta, ki omogoča, da je del programske logike preko injiciranja odvisnosti dostopna vsem drugim komponentam aplikacije. To pomeni, da vsebuje metode in objekte, do katerih dostopa več delov aplikacije, vrednost objekta pa se lahko z uporabo storitve tudi v realnem času spreminja.

- **Krmilnik** je komponenta, ki vsebuje večino programske logike aplikacije na strani odjemalca. Do objektov in metod krmilnikov lahko dostopamo kjerkoli znotraj kode HTML, kjer to primerno definiramo. Izvorna koda 2.4.1 prikazuje primer uporabe krmilnika znotraj dokumenta HTML.

  Izvorna koda 2.4.1: Primer uporabe krmilnika znotraj dokumenta HTML.

```
<div ng-controller="KrmilnikProfila">
    ...
</div>
```

  S tem imamo v sklopu določenega elementa HTML na voljo vse objekte in metode, ki so vezane na `$scope` krmilnika. Zaželeno je, da je programske logike znotraj elementov HTML čim manj.

- **Direktiva** je komponenta, ki predstavlja nek nov element HTML. Poleg tipičnih elementov, kot je `<div></div>`, lahko na primer ustvarimo direktivo `zanka`, ki nek stavek izpiše tolikokrat, kolikor je vrednost podane spremenljivke, uporabimo pa jo kot element `<zanka stevilo=10></zanka>`. Direktivo uporabimo za združevanje kode, ki se v aplikaciji pogosto pojavlja. V njej lahko povežemo izbran krmilnik z izbrano predlogo, ki vsebuje kodo HTML, ki ima potem možnost uporabe celotne funkcionalnosti krmilnika.

- **Pogled** je komponenta, ki vsebuje kodo HTML, ki v glavnem pogledu, `indeks.html` datoteki, nadomesti vrstico `<div ui-view></div>`. Deluje v smislu spletne strani aplikacije, s to razliko, da je pogled vedno le del glavne spletne strani, ki se v našem primeru nikoli ne spremeni. To omogoča izredno hitro menjavo vsebine, ki jo vidi uporabnik aplikacije, saj se glavna spletna stran nikoli ponovno ne naloži, temveč se le zamenja pogled.

AngularJS uporablja princip dvojne izmenjave podatkov, kar omogoča, da se spremenljivka krmilnika v realnem času spreminja tako v kodi HTML kot v samem krmilniku. Spremenljivke so v kodi HTML prikazane na način `{{spremenljivka}}`, kar v brskalniku prikaže njeno vrednost.

2.5 JavaScript in jQuery

JavaScript[13] je objektno usmerjen skriptni jezik in je med najbolj popularnimi programskimi jeziki na svetu. Z vzponom izvajalnega okolja Node.js so nastale tehnologije,

ki omogočajo uporabo JavaScripta na vseh področjih razvoja spletnih aplikacij, tako na strežniku kot pri odjemalcu, zato ni napačno reči, da je JavaScript programski jezik prihodnosti. Glavna zmožnost JavaScripta je implementacija interaktivnosti v aplikacijo, prav tako pa interpreterja zanj ni potrebno posebej nameščati, saj je vključen v večino najbolj popularnih brskalnikov. Ponuja ogromno že razvitih knjižnic, ki omogočajo uporabo funkcionalnosti, katerih delovanje v ozadju nam ni potrebno poznati. jQuery[14] je ena najbolj popularnih knjižnic JavaScript. Glavne prednosti njene uporabe so manjša količina kode, lahkotno upravljanje z elementi HTML, upravljanje dogodkov, animacije in podpora v vseh najbolj popularnih brskalnikih.

2.6 HTML in CSS

HTML[15] je označevalni jezik, ki definira strukturo spletnih dokumentov. To stori z elementi, ki zagotovijo postavitev teksta in slik v brskalniku. Elementi so sestavljeni iz značk, atributov in vsebine: `<značka atribut=vrednost>Vsebina<značka>`. Med tem, ko HTML skrbi za strukturo spletnih dokumentov, CSS[16] skrbi za obliko. To je stilski jezik, ki definira postavitev in izgled elementov HTML. Določa lahko na primer velikost, barvo, ozadje, izgled robov, odmike od drugih elementov, pisave in še marsikaj.

2.7 Git

Git[17] je sistem za upravljanje verzij izvorne kode. Omogoča, da ekipa razvijalcev deluje na istem projektu ob istem času. En član ekipe ima svojo lokalno verzijo projekta, ki jo je prenesel iz strežnika, kjer je naložena najnovejša verzija projekta. Ko član spremeni kodo, lahko novo različico pošlje nazaj na strežnik, od koder si jo prenesejo preostali člani ekipe, ob tem pa so obveščeni o spremembah, tako da lahko prilagodijo svojo različico. Git lahko uporablja tudi posameznik brez interneta na lokalnem računalniku ter si tako prepreči preglavice, ki se zgodijo ob ponesrečeni spremembi kode ali izgubi podatkov.

Poglavje 3 Zamisel

Ideja spletne aplikacije Čutilko je uporabniku ponuditi psihično podporo, kadarkoli in kjerkoli jo potrebuje, če le ima dostop do interneta. Želimo si, da bi si lahko sčasoma uporabnik, ki ga imenujemo čutilec, podporo nudil sam, s pomočjo idej, ki se jih je na naši spletni strani naučil. S tem si bo okrepil samozavest, v trenutku, ko ga preplavijo čustva, pa bo razumel, zakaj se to dogaja in kako se s tem spoprijeti.

V pričakovanju osebnostne rasti mu želimo ponuditi tudi možnost pisanja lastnih idej, ki jih lahko deli z drugimi čutilci, ter tako postane ključni del pri nadaljnjem razvoju vsebine spletne aplikacije. Naš končni cilj je ustvariti neverjetno priložnost osebnostne rasti, h kateri bo s svojimi idejami prispeval sleherni čutilec oziroma uporabnik aplikacije.

Pri idejah je ključno, da jo človek dojame, ne pa da se jo nauči na pamet. Spletna stran mora biti na pogled prijetna in pomirjajoča, da čutilca ne motijo nepotrebni elementi ter je tako sproščen in bolj dovzeten za ideje.

3.1 Zasnova funkcionalnosti

Čutilec po registraciji oziroma vpisu prispe na svojo profilno stran. To je njegov dom, kjer so izpisani njegovi dosežki, misel dneva, dodane ideje, ki jih imenujemo rezine, in nedavne aktivnosti sočutilcev, ki jih spremlja. Sočutilec je nek drug uporabnik aplikacije, ki ga lahko čutilec spremlja tako, da gre na njegovo profilno stran in klikne na gumb »Spremljaj«. Od takrat naprej se na profilni strani čutilca izpiše aktivnost vsakič, ko sočutilec ustvari novo rezino.

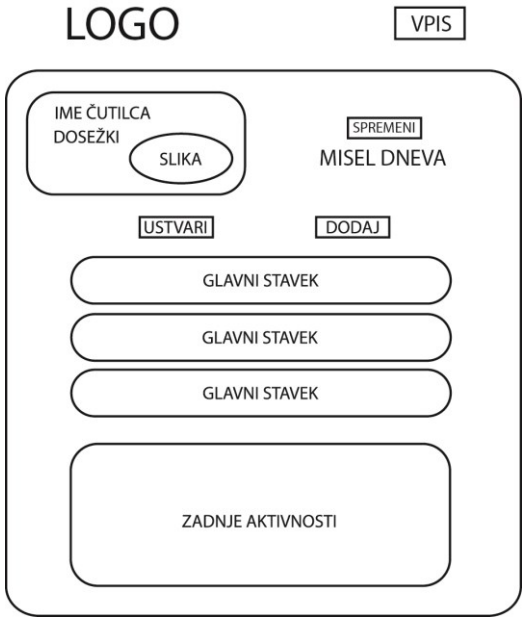
Dosežki so število, ki pove, koliko je čutilec napredoval v smislu osebnostne rasti. Vsakič, ko v svojo zbirko doda rezino, se le-tej poveča moč za eno točko, njemu in avtorju rezine pa se za eno točko povečajo dosežki. Čutilec je tako motiviran, da poleg dodajanja oziroma branja drugih rezin ustvari tudi svoje. S tem, ko si nabira dosežke, prejema različne profilne slike, ki pričajo o njegovem napredovanju.

Misel dneva si čutilec napiše sam in je vidna vsakemu, ki njegov profil obišče. Odraža trenutno stanje čutilca, ki ga želi deliti z drugimi. Pričakujemo, da bodo misli iz dneva v dan bolj pozitivne.

Profil intervalno prikazuje po največ tri dodane rezine naenkrat. Ob vpisu oziroma registraciji v sistem profil vsebuje le eno rezino, ki razloži strukturo rezine in delovanje spletne strani. Ta rezina izgine, ko čutilec doda novo, svojo prvo rezino.

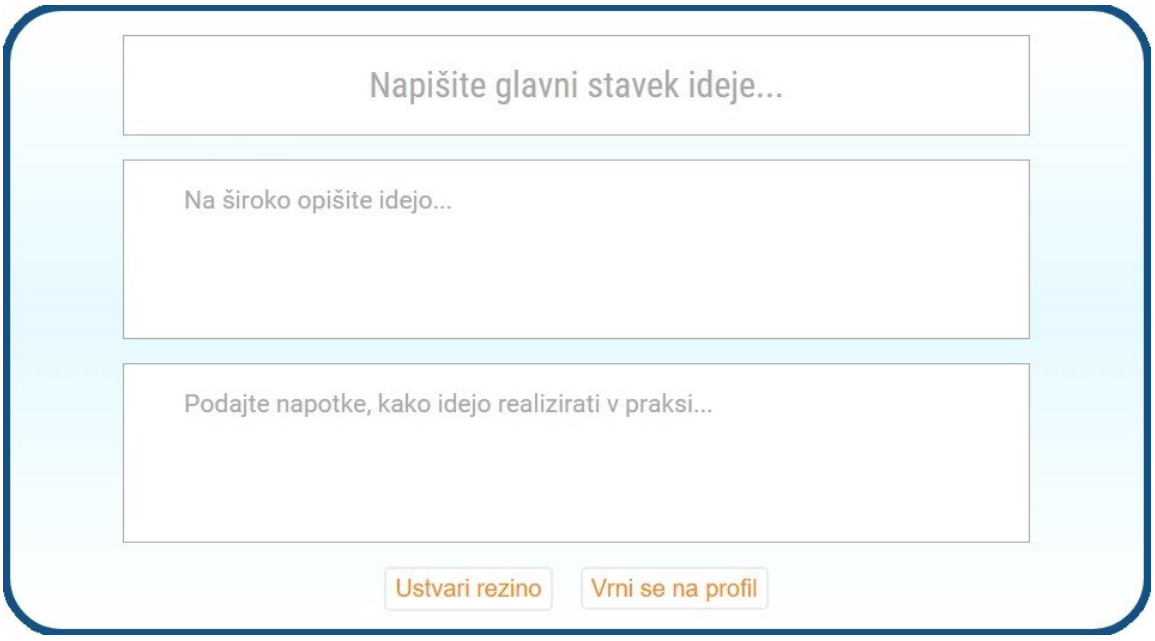
Rezine so sestavljene iz glavnega stavka, jedra in prakse. Glavni stavek je besedilo, ki čutilca spomni na idejo, ki jo rezina opisuje. Biti mora kratek, a dovolj smiselno napisan, da opravi svojo nalogo. V jedru je ideja razložena dovolj podrobno, da v bralcu vzbudi novo razkritje. To je najpomembnejši del rezine, saj bi brez njega imeli le stavek, ki bi morda res nekaj vzbudil v čutilcu, a ta ne bi razumel, zakaj je temu tako. Praksa poda praktične napotke, kako idejo v življenju uresničevati. Brez tega bi čutilec idejo vseeno razumel, a ne bi imela enake vrednosti, saj je morda ne bi znal uporabljati.

Gumb »Dodaj« čutilca popelje na stran za dodajanje, kjer so na voljo vse rezine, ki jih čutilec še nima v svoji zbirki dodanih rezin. Urejene so po moči rezine, ki pove, koliko čutilcev si jo je dodalo v svojo zbirko. Sprva je prikazan le glavni stavek, s klikom nanj pa se rezina odpre in izpiše podrobnosti. Pokažejo se jedro, praksa, ime avtorja, moč rezine in gumb za dodajanje. Ob kliku na ime avtorja rezine se odpre njegova profilna stran, kjer je vidna njegova misel dneva ter vse rezine, ki jih je ustvaril. Slika 3.1.1 prikazuje shematsko zasnovo profilne strani čutilca.



Slika 3.1.1: Shematska zasnova profilne strani čutilca.

Obstaja še stran za ustvarjanje rezin, prikazana na sliki 3.1.2, ki vsebuje obrazec, kjer čutilec vpiše glavni stavek, jedro in prakso. Ko rezino ustvari, je ta na voljo vsem sočutilcem, avtomatsko pa se mu doda tudi v lastno zbirko dodanih rezin.



Slika 3.1.2: Obrazec za ustvarjanje rezin.

Poglavje 4 Podatkovna baza

Preden smo se lotili načrtovanja podatkovne baze smo imeli izkušnje le s sistemom MySQL, ki uporablja medsebojno povezane entitete. Entitetni tip je koncept neke celote z določenimi lastnostmi, ki obstaja v shemi podatkovne baze in je v razmerju z nekim drugim entitetnim tipom, entiteta pa je posamezen primer tega entitetnega tipa. Primer sta avto in človek. Avto ima na primer svojo barvo, končno hitrost in ceno, lastnik pa ime in priimek. Ko konkreten človek kupi konkreten avto, postane lastnik tega avta in entiteti sta v razmerju.

4.1 Entitete

Ker smo se zaradi večje možnosti prilagajanja odločili za sistem MongoDB, je bilo načrtovanje nekoliko drugačno. Posamezni primeri entitetnega tipa tokrat niso imenovani entitete, ampak dokumenti. Shemo je bilo potrebno razviti na drugačen način, a vseeno smo določili entitetne tipe, kot bi to storili v MySQL:

- Edinstvene lastnosti entitetnega tipa **rezina** so čas nastanka (`cas`), `stavek`, `jedro`, `praksa` in moč (`moc`). Vsebuje tudi ključ avtorja (`idCutilca`) in njegovo ime (`imeCutilca`). Ključ avtorja ni neposredno povezan s katerim od drugih dokumentov, lahko pa z njim dokument ročno poiščemo.

- Glavni entitetni tip je **čutilec**. Njegove edinstvene lastnosti so ime čutilca (`imeCutilca`), vrednosti za kodiranje gesla (`hash` in `sol`), dosežki (`dosezki`) in `misel`, vsebuje pa tudi lastnosti, ki so pravzaprav drugi entitetni tipi ali pa njihove zbirke. Entitetni tip čutilec tako vsebuje še lastnosti `ustvarjeno` in `dodano`, ki sta zbirki dokumentov entitetnega tipa rezina, lastnost `zgodovina`, ki je zbirka dokumentov entitetnega tipa aktivnost, in še dve lastnosti `idji_dodanih` ter `socutilci`, ki sta zbirki ključev dodanih rezin in sočutilcev, ki jih čutilec spremlja.

- Edinstvene lastnosti entitetnega tipa **aktivnost** so čas nastanka (`cas`) in `dejanje`, vsebuje pa še lastnosti drugih entitetnih tipov `idCutilca`, `imeCutilca` in objekt entitetnega tipa `rezina`.

Tak pristop smo uporabili, ker MongoDB podatke shranjuje v formatu BSON, ki je binarno predstavljen format JSON. To nam omogoča, da posamezen primer entitetnega tipa shranimo kot dokument z določenimi lastnostmi, te lastnosti pa so lahko tudi posamezni primeri oziroma objekti drugih entitetnih tipov.

4.2 Podatkovna shema

V aplikaciji se vse odvija okrog čutilca. Brez njega ne bi bilo vsebine, zato ima aplikacija le en podatkovni model, `Cutilec`. Znotraj je definirana shema `CutilecSchema`, ki je instanca objekta `mongoose.Schema()` iz modula Mongoose.js. Izvorna koda 4.2.1 prikazuje shemo podatkovnega modela `Cutilec`, ki upošteva vse entitetne tipe.

Izvorna koda 4.2.1: Shema podatkovnega modela `Cutilec`.

```
var CutilecShema = new mongoose.Schema({
  _id: {type: Number, unique: true},
  imeCutilca: {type: String, unique: true},
  hash: String,
  sol: String,
  zgodovina: [{
    cas: { type : Date, default: Date.now },
    idCutilca: Number,
    imeCutilca: String,
    dejanje: String,
    rezina: {
      _id: String,
      idCutilca: Number,
      imeCutilca: String,
      stavek: String,
      jedro: String,
      praksa: String,
      moc: {type: Number, default: 1}
    }
  }],
  dosezki: {type: Number, default: 1},
  ustvarjeno: [{
    _id: String,
    cas: { type : Date, default: Date.now },
    idCutilca: Number,
    imeCutilca: String,
    stavek: String,
    jedro: String,
    praksa: String,
    moc: {type: Number, default: 1}
  }],
  dodano: [{
    _id: String,
    idCutilca: Number,
```

```
      imeCutilca: String,
      stavek: String,
      jedro: String,
      praksa: String,
      moc: {type: Number, default: 1}
    }],
    idji_dodanih: [String],
    misel: {type: String, default: 'Kliknite tu in napiši
te misel, ki se ujema z vašim trenutnim počutjem.'},
    socutilci: [
      {
        _id: Number,
        imeCutilca: String
      }
    ]
});
```

Lastnost `_id` oziroma ključ je avtomatsko dodan vsakemu dokumentu, razen če je posebej definiran. Privzeto je tipa ObjectID, a tudi to lahko spremenimo. V našem primeru je ključ čutilca število, ključ rezine pa niz. Temu je tako, ker za ključ rezine uporabljamo kombinacijo ključa avtorja (čutilca), podčrtaja in zaporedno številko rezine v avtorjevi zbirki ustvarjenih rezin. Nastavljene so tudi nekatere privzete vrednosti lastnosti, na primer stavek za začetno misel in moč rezine ob njenem nastanku. Poleg sheme so definirane tudi njene metode:

- Metoda `nastaviGeslo(geslo)` ustvari naključno vrednost za lastnost `sol`, ki se uporabi pri kodiranju gesla v vrednost lastnosti `hash`.
- Metoda `pravilnoGeslo(geslo)` uporabi že obstoječo vrednost lastnosti `sol` ter podano geslo spet zakodira. Novo dobljeno vrednost `hash` nato primerja z že obstoječo vrednostjo `hash` in vrne `true` ali `false`.
- Metoda `generirajJWT()` ustvari in vrne nov podpisan žeton za avtentikacijo.

Na koncu se z ukazom `mongoose.model('Cutilec', CutilecShema)` model dejansko ustvari in je na voljo v drugih delih aplikacije.

4.3 Seme podatkovne baze

`seme.js` je datoteka, ki prva uporabi model `Cutilec` in se požene le takrat, ko aplikacijo prvič poženemo na strežniku. Ustvari osnovnega čutilca, ki vsebuje vse rezine, ki smo jih ustvarili mi. Podatkovne baze sicer ni potrebno vedno napolniti z začetnimi podatki, a v tem primeru nam to prihrani čas, saj bi morali drugače osnovne rezine vedno znova ročno dodajati.

Preden smo napisali rezine, smo za testno vsebino uporabljali modul Faker, ki omogoča, da namesto pravih vrednosti uporabljamo naključno generirane vrednosti, ki so lahko različnih tipov. Izvorna koda 4.3.1 prikazuje način, kako lahko podatkovno bazo polnimo s testnimi podatki.

Izvorna koda 4.3.1: Sejanje podatkovne baze s testnimi podatki.

```
var Cutilec = mongoose.model('Cutilec');
var faker = require('faker');

var cutilec = new Cutilec();
cutilec._id = 1;
cutilec.imeCutilca = 'Blaž';
cutilec.nastaviGeslo('geslo');
cutilec.ustvarjeno = [{
  _id: '1_1',
  idCutilca: id,
  imeCutilca: cutilec.imeCutilca,
  stavek: faker.Lorem.sentence(),
  jedro: faker.Lorem.paragraph(),
  praksa: faker.Lorem.paragraph()
}];
```

Poglavje 5 Strežniški del

Ob zagonu strežnika se vzpostavi povezava s podatkovno bazo MongoDB, ki avtomatsko ustvari bazo `cutilko`, če ta še ne obstaja, po želji pa lahko izvršimo tudi datoteko `seme.js`, ki podatkovno bazo sprazni in napolni s svežimi podatki. Uvozijo se privzeti moduli, za tem pa se nastavita lokacija za glavni pogled aplikacije in datoteka, ki vsebuje strežniške poti. Prav tako se uvozi modul dotenv, ki nam omogoča, da nastavimo okoljsko spremenljivko `SKRIVNOST`, ki se pozneje uporabi pri avtentikaciji.

5.1 Strežniške poti

V datoteki, ki vsebuje strežniške poti, se najprej uvozijo moduli Mongoose.js, za upravljanje s podatkovno bazo, Passport, za upravljanje z avtentikacijo, JWT za upravljanje z žetoni in Express.js Router, ki omogoča nastavljanje strežniških poti. Poleg tega se uvozi tudi že prej ustvarjeni model `Cutilec`.

Privzeta strežniška pot, prikazana v izvorni kodi 5.1.1, je najbolj enostavna in le poveže osnovni naslov spletne aplikacije z glavnim pogledom `indeks.html`. Definirane ima tip zahteve, spletni naslov in izhodno metodo, ki se izvrši ob uporabi strežniške poti.

Izvorna koda 5.1.1: Privzeta strežniška pot.

```
router.get('/', function(req, res, next) {
  return res.render('indeks');
});
```

Parameter `req` vsebuje podatke, ki jih pošlje odjemalec, `res` pa podatke, ki jih z uporabo ukaza `return` odjemalcu pošlje strežnik nazaj. Naloga parametra `next` je predajanje podatkov naslednjim morebitnih metodam, ki bodo podatke obdelovale. Tako izgleda vsaka strežniška pot, le da imajo lahko drugačno zahtevo in spletni naslov, v sami metodi pa se izvrši še več programske logike, preden metoda vrne neko vrednost.

V datoteki je poleg strežniških poti definirana še metoda `je_id_v_tabeli(id, tabela)`, ki preveri, ali je podan ključ v podani tabeli, in pa strežniška pot s posebno

zahtevo PARAM, ki vsakič, ko spletni naslov vključuje ključ čutilca, vnaprej pridobi njegove podatke, kar prikazuje izvorna koda 5.1.2.

Izvorna koda 5.1.2: Zahteva PARAM, ki vrne podatke o čutilcu.

```
router.param('cutilec', function(req, res, next, id) {

  Cutilec.findOne({
    _id: id
  }, function(err, cutilec) {
    if (err) {
      return next(err);
    }
    if (!cutilec) {
      return next(new Error('Na najdem čutilca.'));
    }
    req.cutilec = cutilec;
    return next();
  });
});
```

Strežnik vsebuje še naslednje strežniške poti:

- Pot z zahtevo GET na naslov `/api/v1/cutilci/:cutilec`, ki uporabi parameter `cutilec` iz naslova, da iz podatkovne baze potegne pravega čutilca ter ga vrne odjemalcu.

- Pot z zahtevo POST na naslov `/api/v1/rezine`, ki preko `req` dobi ključ čutilca in rezino. To rezino doda v čutilčevo tabelo dodanih rezin v podatkovni bazi in mu poveča dosežke za eno točko. Prav tako za eno točko poveča moč rezine in dosežke avtorja rezine. Odjemalcu vrne potrditev o uspehu.

- Pot z zahtevo GET na naslov `/api/v1/rezine/:cutilec`, ki uporabi parameter `cutilec` iz naslova, da iz podatkovne baze potegne pravega čutilca in odjemalcu vrne vse rezine, ki jih ta čutilec še nima v svoji zbirki dodanih rezin.

- Pot z zahtevo POST na naslov `/api/v1/rezine/:cutilec`, ki uporabi parameter `cutilec` iz naslova, da iz podatkovne baze potegne pravega čutilca. Preko `req` dobi rezino, ki jo je ta čutilec ustvaril, in jo doda v njegovo tabelo ustvarjenih rezin v podatkovni bazi. Prav tako jo doda v njegovo zbirko dodanih rezin, mu poveča dosežke za eno točko in v tabelo `zgodovina` doda novo aktivnost. Ta aktivnost vsebuje informacije o tem, kdaj je uporabnik ustvaril rezino in na katero rezino se to navezuje. Odjemalcu vrne potrditev o uspehu.

- Pot z zahtevo PUT na naslov `/api/v1/misel/:cutilec`, ki uporabi parameter `cutilec` iz naslova, da iz podatkovne baze potegne pravega čutilca. Preko `req` dobi novo misel, s katero v podatkovni bazi prepiše staro misel. Odjemalcu vrne potrditev o uspehu.

- Pot z zahtevo GET na naslov `/api/v1/zgodovina/:cutilec`, ki uporabi parameter `cutilec` iz naslova, da iz podatkovne baze potegne pravega čutilca in odjemalcu vrne zgodovino vseh drugih uporabnikov oziroma sočutilcev, ki jih ima v tabeli `spremlja`.

- Pot z zahtevo POST na naslov `/api/v1/socutilci/:cutilec`, ki uporabi parameter `cutilec` iz naslova, da iz podatkovne baze potegne pravega čutilca. Preko `req` dobi sočutilca, ki ga doda v čutilčevo tabelo dodanih sočutilcev. Odjemalcu vrne potrditev o uspehu.

- Pot z zahtevo POST na naslov `/api/v1/registracija`, ki preko `req` dobi ime in geslo čutilca. V kolikor eden od njiju manjka, vrne napako. Za tem preveri, koliko čutilcev je že v podatkovni bazi in novemu čutilcu dodeli ključ, ki je za ena večji od ključa zadnjega čutilca v podatkovni bazi, nato pa ga shrani v podatkovno bazo. Odjemalcu vrne žeton, ki ga generira s pomočjo metode `generirajJWT()` iz modela `Cutilec`.

- Pot z zahtevo POST na naslov `/api/v1/vpis`, ki preko `req` dobi ime in geslo čutilca. V kolikor eden od njiju manjka, vrne napako. Za tem uporabi metodo `authenticate()` iz modula Passport, ki preveri ustreznost podatkov, in odjemalcu vrne žeton, ki ga generira s pomočjo metode `generirajJWT()` iz modela `Cutilec`.

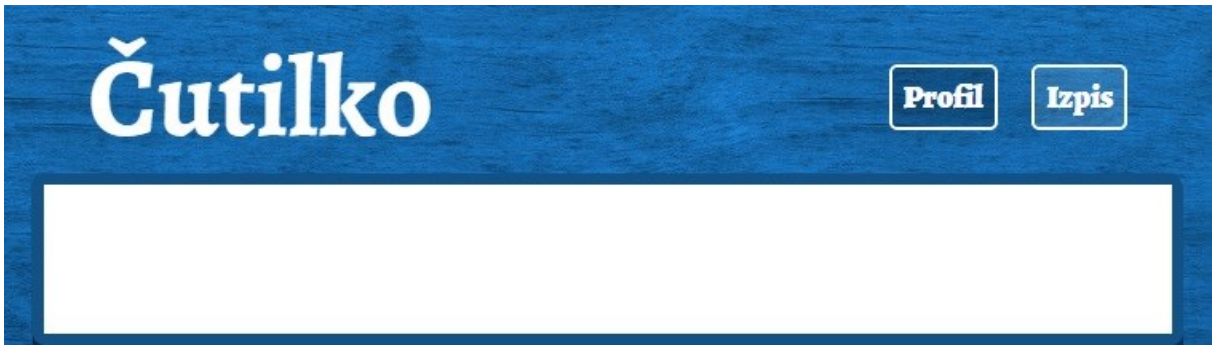
5.2 Glavni pogled

Glavni pogled vsebuje kodo HTML, ki je v aplikaciji vedno vidna. Deluje kot okvir za poglede ogrodja AngularJS, ki se znotraj glavnega pogleda izmenjujejo glede na zahteve čutilca. V prvi vrstici kode je definiran modul ogrodja AngularJS, ki naj ga glavni pogled uporablja. Mi smo ustvarili modul `cutilko`, kar definiramo takole: `<html ng-app="cutilko">`. Vse komponente AngularJS, ki so povezane z modulom `cutilko`, so sedaj na voljo za uporabo znotraj kode HTML.

Znotraj glavnega pogleda je definiranih še več stvari:

- Za predstavitev teksta uporablja aplikacija kombinacijo pisav Alegreya, Roboto Condensed in Roboto.
- Poleg po meri napisane kode CSS uporablja aplikacija tudi datoteko `normalize.css`, ki nastavi temeljne stilske lastnosti elementov HTML, kar ponuja konstanten začetek, preden se sami lotimo oblikovanja.
- Vse poti do datotek oziroma skript, ki jih uporablja aplikacija za poganjanje interaktivnosti, so definirane na koncu. To vključuje tudi vse po meri narejene komponente AngularJS.

Glavni del je zelo kratek in vključuje le naslovno vrstico z navigacijskimi gumbi ter belo podlago z robovi, kjer se prikazujejo pogledi ogrodja AngularJS. Slika 5.2.1 prikazuje glavni pogled brez vstavljenega pogleda AngularJS.



Slika 5.2.1: Glavni pogled aplikacije brez vstavljenega pogleda AngularJS.

Poglavje 6 Odjemalni del

6.1 Storitve

AngularJS ima že ob namestitvi na voljo ogromno storitev, ki so pripravljene za takojšnjo uporabo. Tiste, ki smo jih največkrat uporabili, so `$http` za pošiljanje spletnih zahtev, `$state` za upravljanje trenutnega pogleda, `$interval` za delo s časovnimi intervali in `$scope` za delo z objekti in metodami krmilnikov.

Med razvijanjem smo odkrili potrebo po treh dodatnih storitvah:

- Storitev **avten** je namenjena upravljanju žetonov, vpisa, izpisa in registracije čutilcev ter pridobivanju pomembnih informacij o čutilcu.
- Storitev **rezine** deluje kot vmesnik med strežnikom in odjemalcem, ki upravlja z rezinami.
- Storitev **cutilci** deluje kot vmesnik med strežnikom in odjemalcem, ki upravlja s čutilci.

6.2 Krmilniki

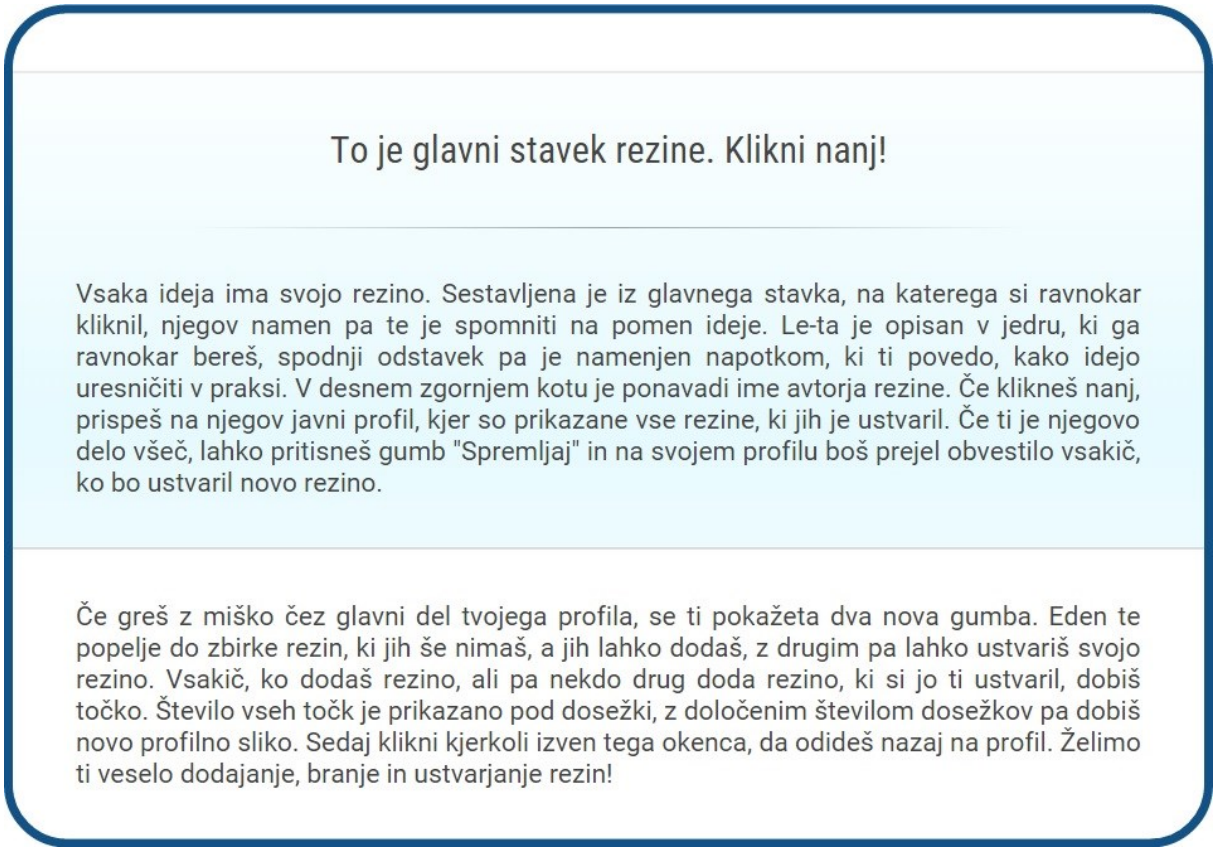
Med razvijanjem smo ustvarili štiri krmilnike:

- **KrmilnikNavigacije** ja najbolj enostaven. Deluje le kot posrednik med elementi HTML in storitvijo **avten**. Izvorna koda 6.2.1 prikazuje primer enostavnega krmilnika, ki s pomočjo storitve **avten** napolni svoje spremenljivke.

Izvorna koda 6.2.1: Primer enostavnega krmilnika.

```
angular.module('cutilko')
  .controller('KrmilnikNavigacije', ['$scope', 'avten',
 function($scope, avten) {
    $scope.jeVpisan = avten.jeVpisan;
    $scope.izpis = avten.izpis;
    $scope.trenCutilec = avten.trenCutilec;
  }]);
```

- **KrmilnikAvtentikacije** je potreben za delovanje logike vpisa in registracije.
- **KrmilnikRezin** skrbi za programsko logiko prikaza, dodajanja in ustvarjanja rezin na strani odjemalca.
- **KrmilnikProfila** služi upravljanju programske logike delovanja profilne strani. Definirano ima tudi začetno rezino, prikazano na sliki 6.2.1, ki čutilcu razloži strukturo rezin in delovanje aplikacije, vidna pa je samo dokler čutilec ne doda nove, svoje prve rezine.



Slika 6.2.1: Začetna rezina, ki čutilcu razloži delovanje aplikacije.

6.3 Direktive

Med razvojem se je pokazala potreba po dveh direktivah:

- Direktiva **rezina** je najpomembnejši element celotne aplikacije. Čeprav vsebuje izredno malo kode, ki je prikazana v izvorni kodi 6.3.1, med seboj povezuje programsko logiko iz krmilnika rezin in predlogo `rezina.html`, kar omogoča, da kodo HTML iz predloge uporabimo neštetokrat, ta pa lahko uporablja polno

funkcionalnost krmilnika. Ustvarjamo torej instance predloge HTML, kjer lahko vsaka prikazuje drugačno vsebino, čeprav je koda HTML enaka. To je možno, ker ob izvršitvi posameznim instancam podamo objekte, ki vsebujejo edinstveno vsebino, na primer takole: `<imeDir imeObjvDir=podaniObjekt></imeDir>`.

Izvorna koda 6.3.1: Primer enostavne direktive.

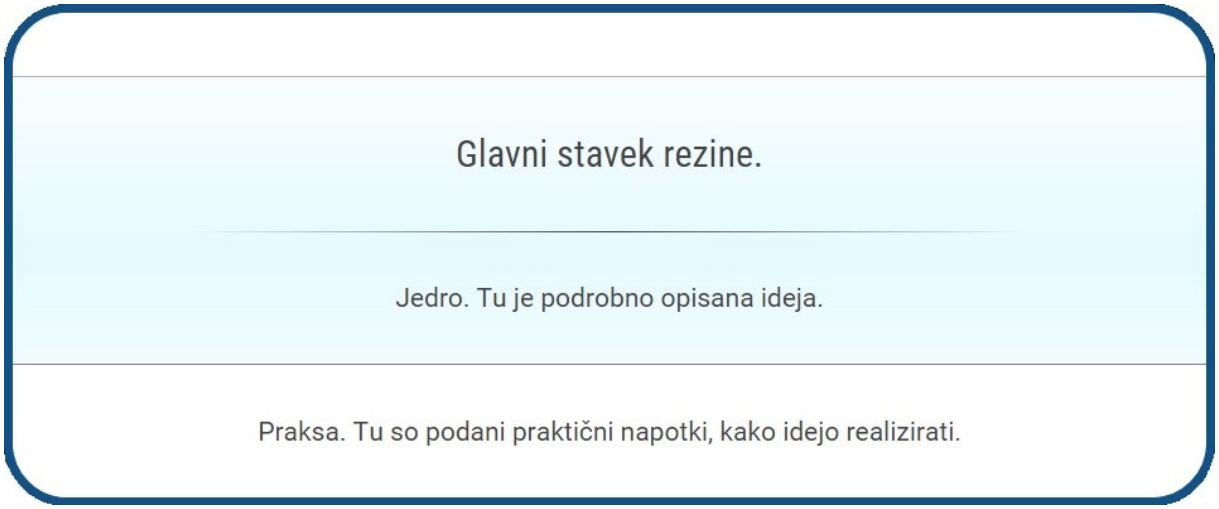
```
angular.module('cutilko')
  .directive('rezina', function() {
    return {
      restrict: 'E',
      transclude: true,
      scope: {
        rezina: '='
      },
      templateUrl: 'predloge/rezina.html',
      controller: 'KrmilnikRezin'
    };
  });
```

- Direktiva **misel** je nastala zaradi potrebe po spreminjanju besedila elementa `<p>` v kodi HTML. Že obstoječe rešitve nismo našli, zato smo ustvarili svojo. Direktiva upravlja z vrednostmi in prikazovanjem treh elementov HTML, odstavka `<p>`, prostora za pisanje `<textarea>` in gumba `<button>`. Ob spremembi besedila odstavka poskrbi, da se novo besedilo shrani v podatkovno bazo.

6.4 Predloga rezina

Predloga je zbirka kode HTML, ki jo uporabljajo direktive, v našem primeru direktiva **rezine**. Bistvo predloge `rezina` je ta, da nam ni potrebno vedno znova pisati kode za koncept rezine. Pričakujemo namreč, da se bo skozi čas v aplikaciji nabralo ogromno rezin, zato je bilo potrebno ustvariti primerno predlogo.

Uporablja krmilnik rezin, ki vsebuje kar nekaj spremenljivk tipa Boolean, ki določajo, kateri elementi rezine so vidni in kateri ne. Rezina je na začetku zaprta, kar pomeni, da je viden le glavni stavek in v primeru, da smo na pogledu za dodajanje rezin, tudi moč. Ko kliknemo na rezino, se le-ta odpre na sredini zaslona, vsebina v ozadju pa izgine, tako da je vidna le odprta rezina. Ta sedaj prikazuje glavni stavek, jedro, prakso in ime avtorja, v primeru, da smo na pogledu za dodajanje rezin, pa še moč in gumb za dodajanje rezine v zbirko čutilčevih dodanih rezin. Na sliki 6.4.1 je vidna struktura rezine.



Slika 6.4.1: Struktura rezine.

6.5 Pogledi

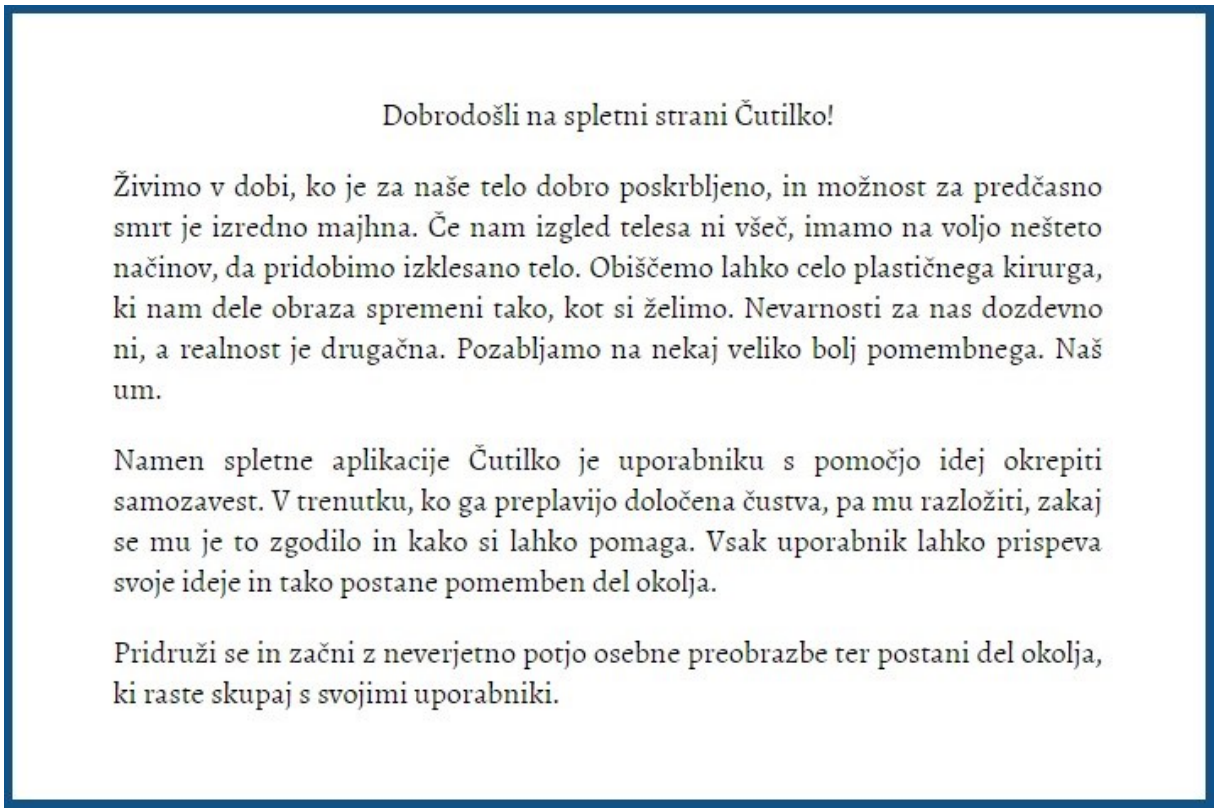
AngularJS uporablja odjemalne poti, ki povedo, kateri pogled naj aplikacija odpre, ko čutilec dostopa do določenega spletnega naslova. Vsi pogledi imajo svojo odjemalno pot z enakim imenom, povezani pa so tudi z določenim krmilnikom. Izvorna koda 6.5.1 prikazuje primer odjemalne poti. Odjemalne poti dopuščajo tudi možnost, da se vsakič, ko se zamenja pogled, izvrši določena koda, ki vnaprej pripravi podatke za ta pogled ali pa na primer preveri, če je čutilec vpisan in sploh ima pravico do vstopa v ta pogled.

Izvorna koda 6.5.1: Primer odjemalne poti.

```
.state('rezine', {
      url: '/rezine',
      templateUrl: '/pogledi/rezine/indeks.html',
      controller: 'KrmilnikRezin',
      resolve: {
        obljubaRezin: ['rezine', function(rezine) {
          return rezine.dobiRezine();
        }]
      },
   onEnter: ['$state', 'avten', function($state, avten)
      {
        if (!avten.jeVpisan()) {
          $state.go('vstopna');
        }
      }]
```

6.5.1 Vstopni pogled

Vstopni pogled, prikazan na sliki 6.5.1.1, je viden vsem, ki obiščejo spletno aplikacijo, in ne uporablja nobenih drugih funkcionalnosti ogrodja AngularJS. Vsebuje stavek za pozdrav in nekaj odstavkov, ki dajo obiskovalcu idejo o tem, čemu aplikacija služi. To se nam zdi pomembno, saj smo mnenja, da se bo redko kdo registriral v aplikaciji, katere namen ne pozna.



Slika 6.5.1.1: Vstopni pogled.

6.5.2 Pogleda za vpis in registracijo

Pogleda za vpis in registracijo, prikazana na slikah 6.5.2.1 in 6.5.2.2, sta povezana s krmilnikom avtentikacije. Ta vsebuje prazen objekt `cutilec`, ki se napolni z vrednostmi, pridobljenimi iz spletnih obrazcev. Ko čutilec izpolni polji za ime in geslo čutilca ter pritisne na gumb za vpis oziroma registracijo, se najprej izvrši metoda `velikaZacetnica(ime)`, ki spremeni podano ime v format, kjer je prva črka velika črka, vse druge pa male. Za tem se izvrši še glavna metoda `vpis()` oziroma `registracija()`. Ta izvrši enako poimenovano metodo iz storitve **avten** ter zraven kot argument poda objekt `cutilec`, kar dejansko vpiše oziroma registrira čutilca ter ga preusmeri na pogled njegovega profila. Če se

pri procesu pojavi težava, se v krmilniku ustvari nova spremenljivka `napaka`, ki se uporabi pri obveščanju čutilca o neuspeli avtentikaciji.

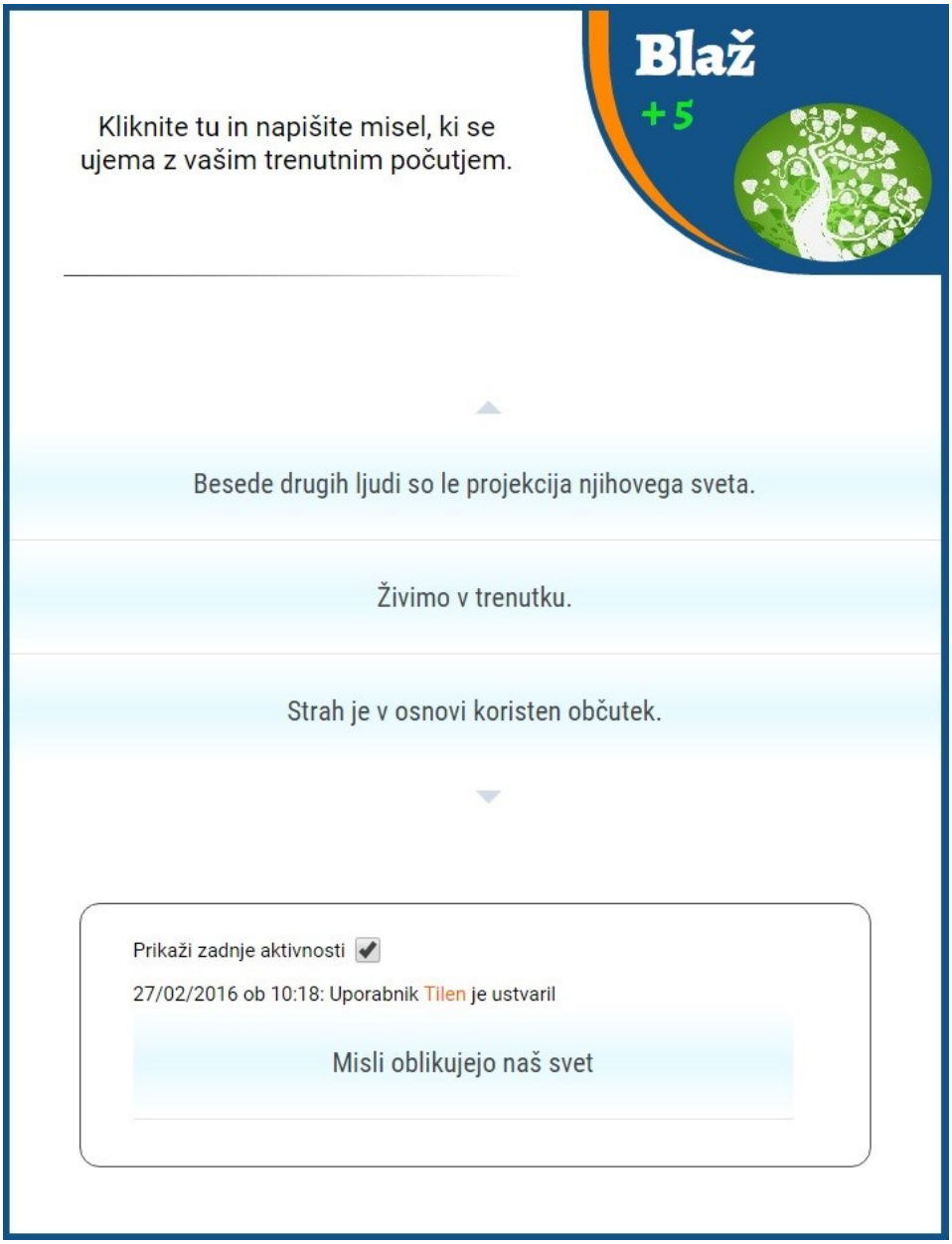
Polje za pisanje imena je omejeno na minimalno tri in maksimalno pet črk ter ob praznem polju prikazuje besedilo »Ime Čutilca«. Polje za pisanje gesla nima posebnih omejitev, ob praznem polju pa prikazuje besedilo »Geslo«. V primeru, da čutilec še ni registriran, ima v pogledu za vpis na voljo povezavo do pogleda za registracijo.

Slika 6.5.2.1: Obrazec za vpis.

Slika 6.5.2.2: Obrazec za registracijo.

6.5.3 Pogled profila

Ob preklopu na pogled profila, ki je povezan s krmilnikom profila, se najprej izvrši `obljubaProfila`. Ta izvrši metodo `dobiProfil(null)` iz storitve **cutilci**, ki pošlje GET zahtevo, s katero strežnik vrne vse podatke trenutnega čutilca. Krmilnik profila nato uporabi te podatke, da napolni svoje spremenljivke. Če obiščemo profil sočutilca, se namesto parametra `null` uporabi ključ sočutilca, kar izvrši drugo kodo metode in vrne podatke za sočutilca. Slika 6.5.3.1 prikazuje primer pogleda profila čutilca.



Slika 6.5.3.1: Primer pogleda profila čutilca.

Profil je sestavljen iz štirih glavnih elementov:

- **Misel** zaseda približno 60% prve vrste elementov in je instanca direktive **misel**. Odstavek znotraj direktive prikazuje vrednost spremenljivke `profil.misel` iz krmilnika profila. Ob kliku nanj se odstavek skrije, prikaže pa se polje za urejanje besedila odstavka ter gumb za shranjevanje. Ko besedilo spremenimo in kliknemo na gumb, se izvrši metoda `spremeniMisel()` iz storitve **cutilci**, ki pošlje PUT zahtevo skupaj z novo mislijo, ki v podatkovni bazi zamenja staro.

- **Profilka** zaseda preostali del prve vrste elementov in prikazuje ime čutilca, njegove dosežke in pa sliko profila. S pomočjo CSSa smo ustvarili modro podlago in pa oranžno krivuljo, ki poudari mejo med profilko in ostalimi elementi, kot je to vidno na sliki 6.5.3.1 zgoraj desno. Dosežki so napisani v zeleni barvi, ker želimo, da predstavljajo nekaj pozitivnega. Slika za svoj vir (atribut `src` v znački `<img>`) ne uporablja direktno niza, temveč uporablja niz, ki ga vrne metoda `dobiSliko()`. Ta preveri število dosežkov čutilca in za določeno število vrne niz lokacije določene slike. S tem čutilec dobiva vedno bolj navdušujoče slike.

- **Jedro profila** zaseda celotno naslednjo vrsto elementov. V krmilniku profila je tabela `pogledRezin`, ki dobi tri rezine iz tabele `profil.dodano` in jih s pomočjo direktive **rezine** prikazuje na profilu. Te rezine se izmenjujejo na več načinov. Krmilnik ima metodi `gor()` in `dol()`, ki tabelo `pogledRezin` napolnita z novimi največ tremi rezinami, ki so v tabeli `profil.dodano` na enem indeksu nižje oziroma višje, odvisno od metode. Če kliknemo na gumb nad jedrom profila, se izvrši prva, če kliknemo na gumb pod jedrom pa druga metoda. Gumba sta aktivna le, ko imamo dodane več kot tri rezine. V kolikor se ne vmešavamo, se s pomočjo intervala vsake pol minute samodejno požene metoda `dol()`. Ko se z miško pomaknemo čez jedro profila, se prikažeta tudi gumba za preklop na poglede dodajanja in ustvarjanja rezin.

- **Zgodovina** zaseda celotno zadnjo vrsto elementov in je prikazana le, če tabela `profil.zgodovina` ni prazna. Če je temu tako, lahko s pomočjo gumba določimo, ali bomo podatke prikazovali. Tabela vsebuje zadnje aktivnosti vseh sočutilcev, ki jih spremljamo. Aktivnost vsebuje čas, ko je bila aktivnost zabeležena, ime sočutilca, na katerega se navezuje, kakšno je dejanje sočutilca in katera rezina je z aktivnostjo povezana. Urejene so po času padajoče, prikazuje se le zadnjih pet, rezine pa so znova predstavljene z direktivo **rezine**. Če kliknemo na ime sočutilca, nas to popelje na njegovo profilno stran.

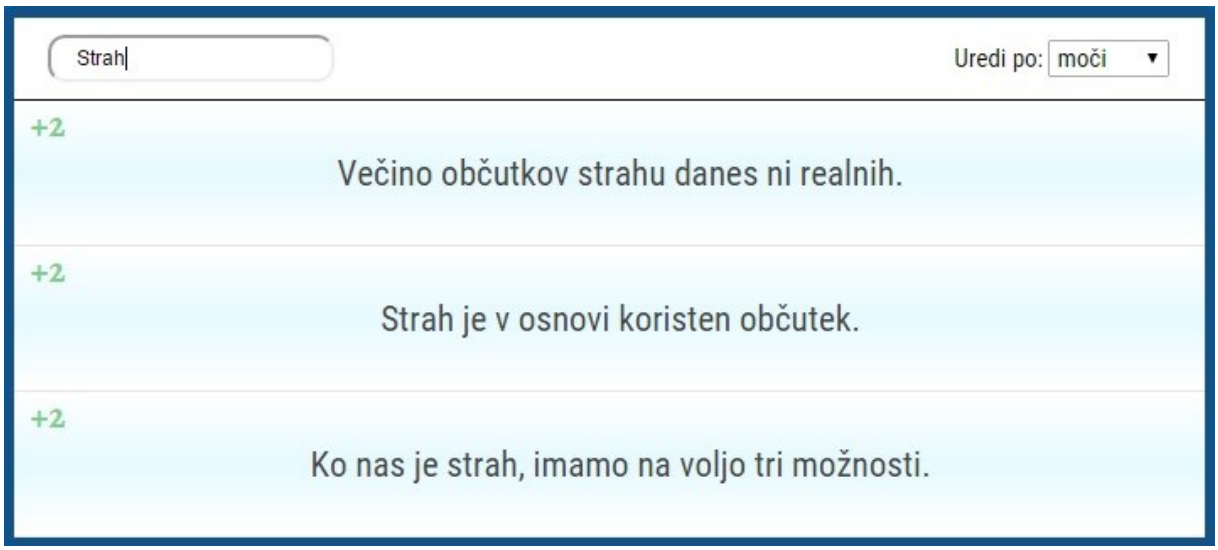
6.5.4 Pogled rezin

Ob preklopu na pogled rezin, ki je povezan s krmilnikom rezin, se najprej izvrši `obljubaRezin`. Ta izvrši metodo `dobiRezine()` iz storitve **rezine**, ki pošlje GET zahtevo, kot je prikazano v izvorni kodi 6.5.4.1, s katero strežnik vrne vse rezine, ki jih čutilec še nima v zbirki dodanih rezin. Lastnost `headers` je uporabljena za preverjanje pristnosti čutilca na strani strežnika.

Izvorna koda 6.5.4.1: Primer $http zahteve GET.

```
r.dobiRezine = function() {
    return $http.get('/api/v1/rezine/' + avten.idTrenCu
tilca(), {
      headers: {
        Authorization: 'Bearer ' + avten.dobiZeton()
      }
    }).success(function(rezine) {
      angular.copy(rezine, r.rezine);
    });
  };
```

Pogled, prikazan na sliki 6.5.4.1, ima na vrhu polje, kjer lahko iščemo rezine, ki vsebujejo določene besede, in meni, kjer izbiramo med urejenostjo rezin glede na moč ali čas nastanka. Od tu dalje so prikazane vse rezine, ki jih uporabnik še nima dodanih. Vidita se le moč in glavni stavek rezine, kar pomeni, da so rezine zaprte. Če katerikoli stavek v nas vzbudi zanimanje, lahko nanj kliknemo in rezina se odpre. To se zgodi z izvršitvijo metode `odpri()`, ki nastavi določeno Boolean spremenljivko na vrednost `true`, kar povzroči, da se rezini spremeni stil prikaza. Vidni postanejo še jedro, praksa, ime avtorja in pa gumb za dodajanje. Ob kliku na ime avtorja preklopimo na njegov profil, ob kliku na gumb za dodajanje pa se izvrši metoda `dodajRezino(rezina)`, ki rezino doda v zbirko dodanih rezin ter jo odstrani iz seznama ne dodanih. Ob kliku v prazno polje poleg odprte rezine se leta zapre.



Slika 6.5.4.1: Pogled za dodajanje rezin.

6.5.5 Pogled ustvarjanja rezin

Pogled ustvari, prikazan na sliki 3.1.2, s pomočjo obrazca napolni objekt `novaRezina` s tremi lastnostmi:

1. V prvo polje vpišemo glavni stavek ideje, ki napolni lastnost `novaRezina.stavek`. Minimalna dolžina besedila je deset črk, maksimalna pa petdeset. Ko je polje prazno je izpisano besedilo »Napišite glavni stavek ideje…«.

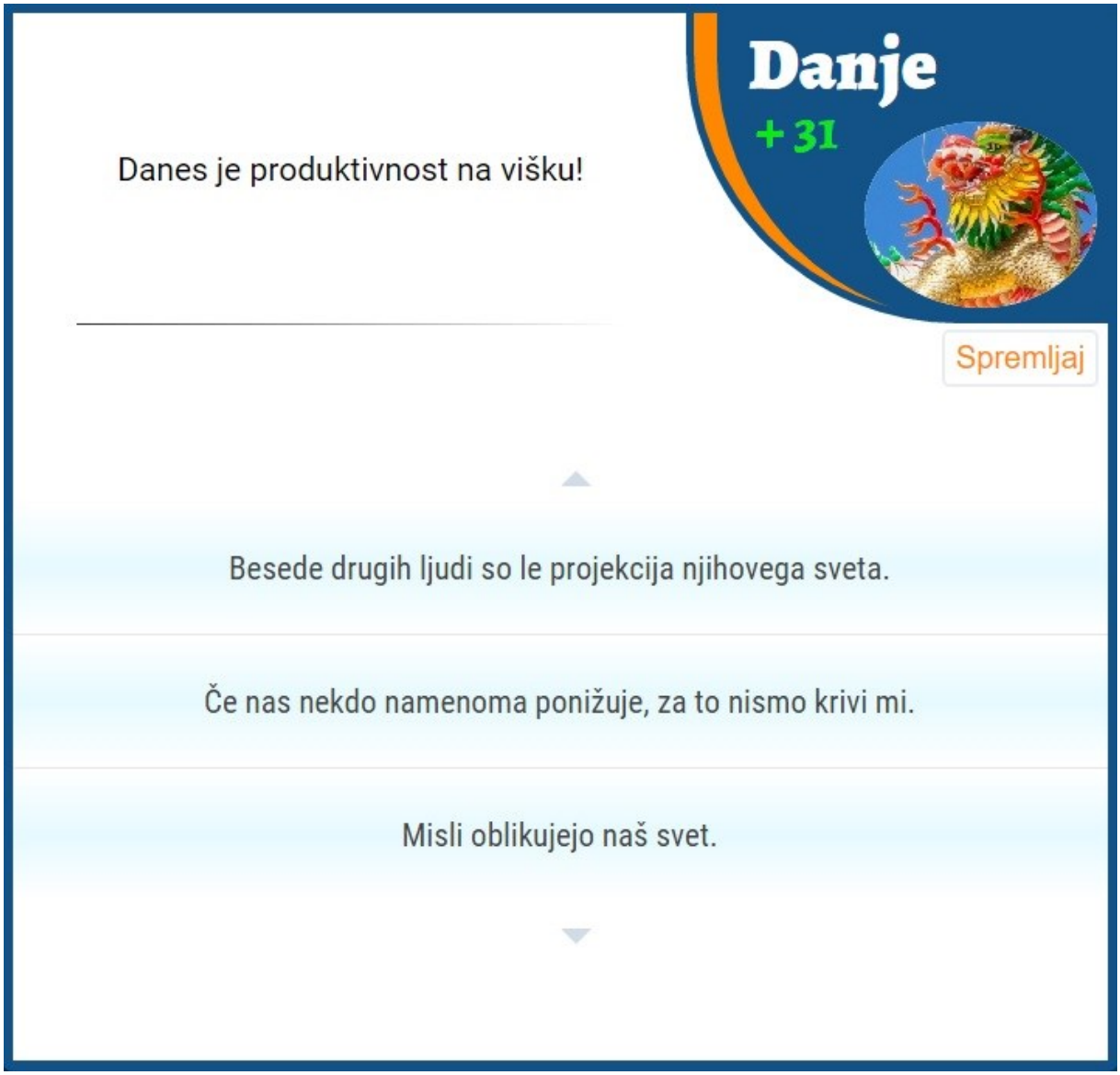
2. V drugo polje vpišemo jedro rezine, ki napolni lastnost `novaRezina.jedro`. Minimalna dolžina besedila je sto črk, maksimalna pa petsto. Ko je polje prazno je izpisano besedilo »Na široko opišite idejo…«.

3. V zadnjem polju vpišemo prakso rezine, ki napolni lastnost `novaRezina.praksa`. Minimalna dolžina besedila je petdeset črk, maksimalna pa tristo. Ko je polje prazno je izpisano besedilo »Podajte napotke, kako idejo realizirati v praksi…«.

Vsa tri polja morajo biti obvezno izpolnjena, da se rezina lahko doda. Ko kliknemo na gumb »Ustvari rezino«, se izvrši metoda `ustvariRezino()` iz krmilnika rezin, ki objektu `novaRezina` doda še ključ in ime trenutnega čutilca. Za tem uporabi enako poimenovano metodo storitve **rezina** ter tako ustvari novo rezino v podatkovni bazi. Če rezine ne želimo ustvariti, lahko kliknemo gumb, ki nas vrne nazaj na pogled profila.

6.5.6 Pogled profila sočutilca

Pogled profila sočutilca je prikazan na sliki 6.5.6.1 in je skoraj popolnoma enak pogledu profila, le prirejen je za sočutilca. Odstranjena je možnost spreminjanja misli in izpis zgodovine oziroma zadnjih aktivnosti. Prav tako se v jedru profila ne izpisujejo dodane rezine sočutilca, ampak tiste, ki jih je ustvaril. Dodan je gumb »Spremljaj«, ki izvrši metodo `dodajSocutilca()`, ki sočutilca doda v zbirko sočutilcev, ki jih čutilec spremlja.

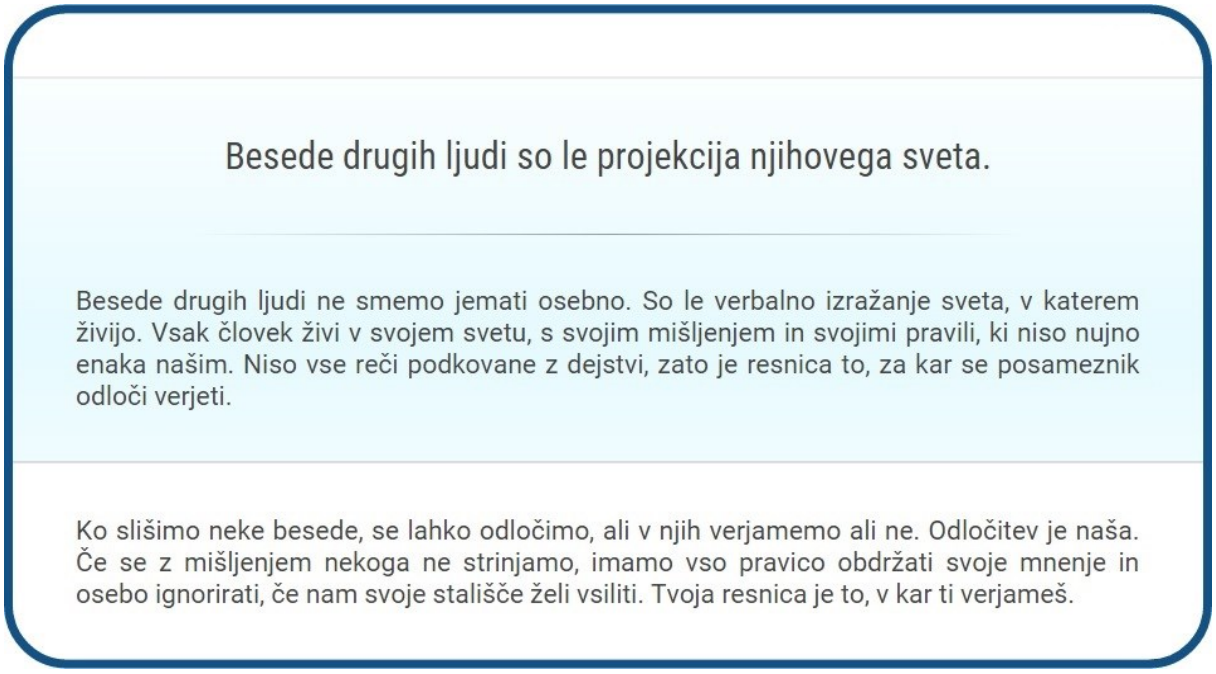


Slika 6.5.6.1: Profilna stran sočutilca.

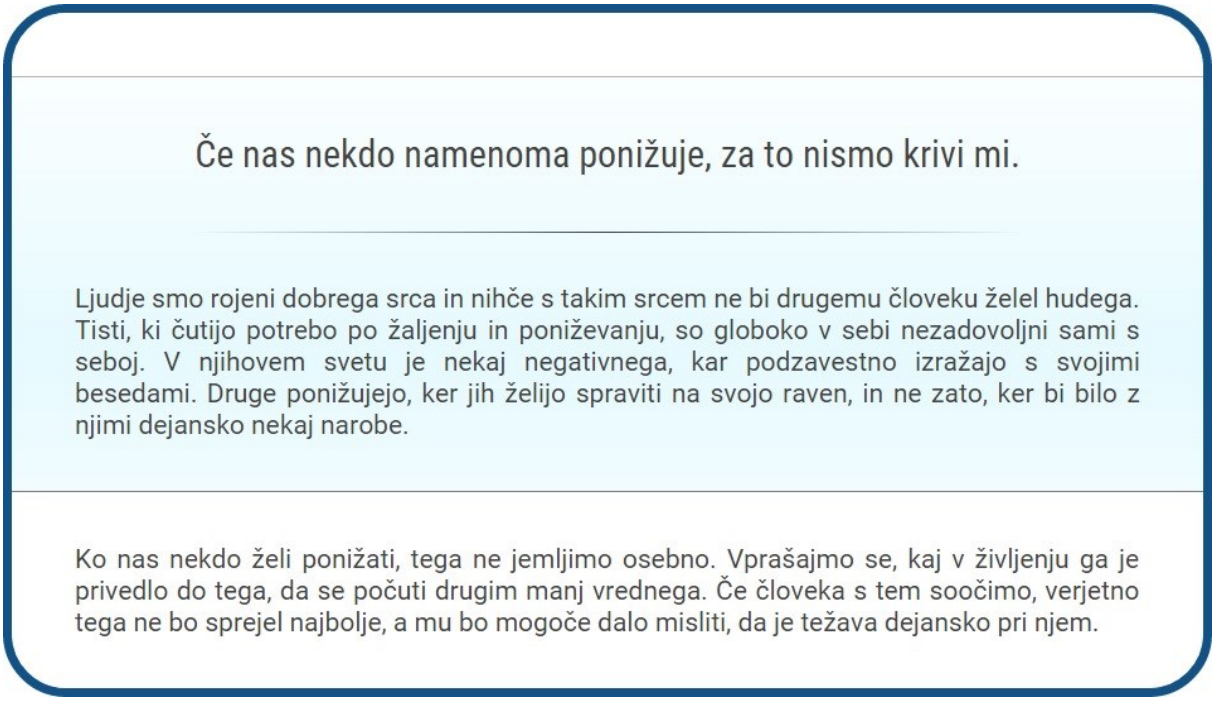
6.6 Osnovne rezine iz aplikacije

Osnovne rezine, ki smo jih vključili v aplikacijo, so:

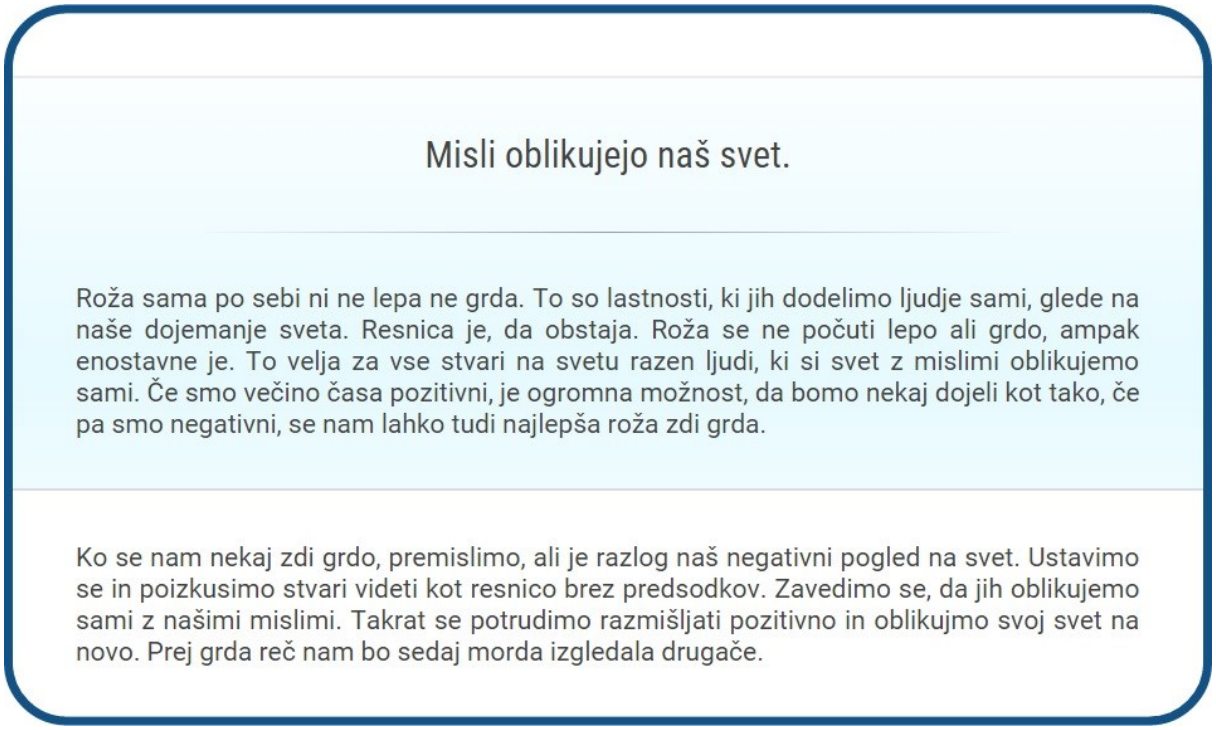
1. Besede drugih ljudi so le projekcija njihovega sveta: slika 6.6.1.
2. Če nas nekdo namenoma ponižuje, za to nismo krivi mi: slika 6.6.2.
3. Misli oblikujejo naš svet: slika 6.6.3.
4. Zaupati moramo lastni presoji in na stvari gledati z lastnimi očmi: slika 6.6.4.
5. Bodimo glavna zvezda našega življenja: slika 6.6.5.
6. Živimo v trenutku: slika 6.6.6.
7. Pohvala nima vedno željenega učinka: slika 6.6.7.
8. Fizično smo enakovredni drugim: slika 6.6.8.
9. Značaj se oblikuje skozi odraščanje: slika 6.6.9.
10. Ljudje se lahko ustrašijo samozavesti: slika 6.6.10.
11. Ne postanimo odvisni od tehnologije: slika 6.6.11.
12. Materialne dobrine ne zapolnijo praznine: slika 6.6.12.
13. Ideja dobi pravo vrednost šele takrat, ko jo pričnemo uresničevati: slika 6.6.13.
14. Brez zadržkov izražajmo svojo osebnost in ne glejmo na posledice: slika 6.6.14.
15. Ne bojimo se delati napak: slika 6.6.15.
16. Sposobni moramo biti živeti sami: slika 6.6.16.
17. Od sebe ne pričakujmo popolnosti: slika 6.6.17.
18. Večina strahov danes ni realnih: slika 6.6.18.
19. Strah je v osnovi koristen občutek: slika 6.6.19.
20. Ko nas je strah, imamo na voljo tri možnosti: slika 6.6.20.
21. Občutek strahu je tak z razlogom: slika 6.6.21.
22. Dihanje je kratek stik za paniko: slika 6.6.22.
23. V trenutku panike uporabimo pomirjevalni pristop: slika 6.6.23.



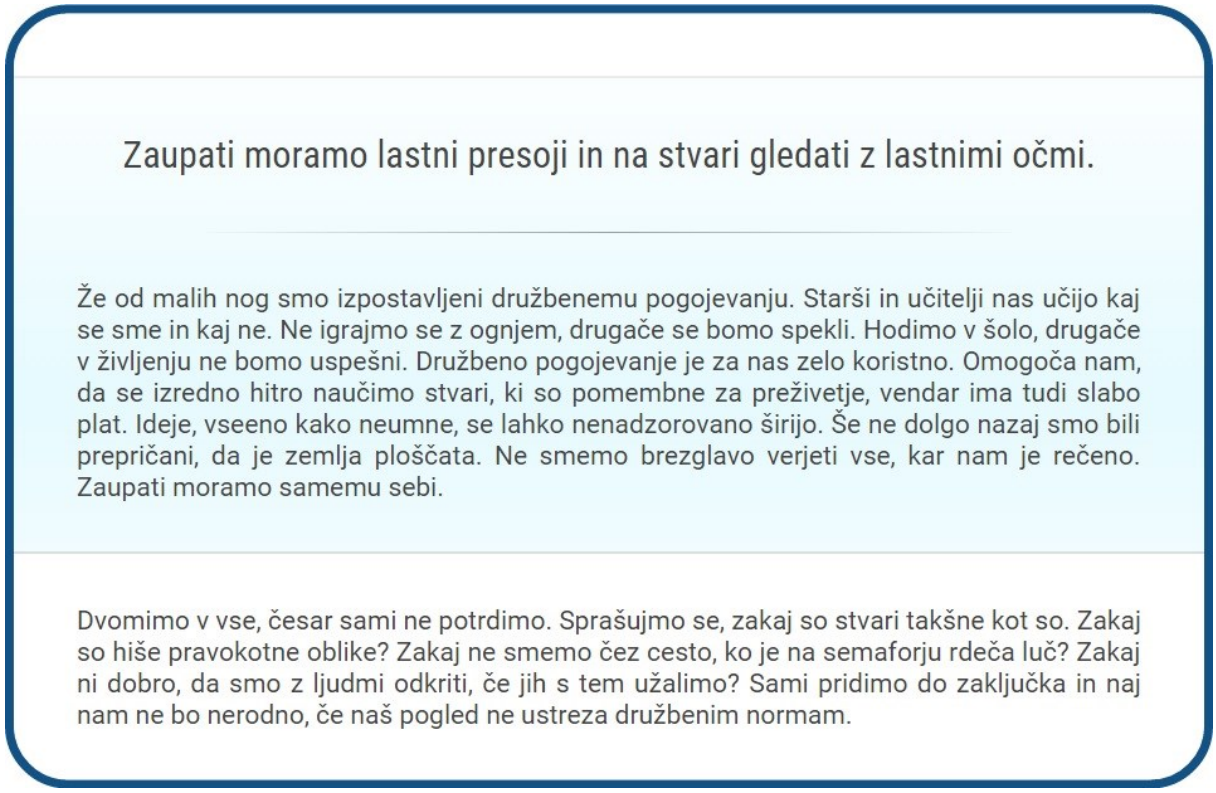
Slika 6.6.1: Besede drugih ljudi so le projekcija njihovega sveta.



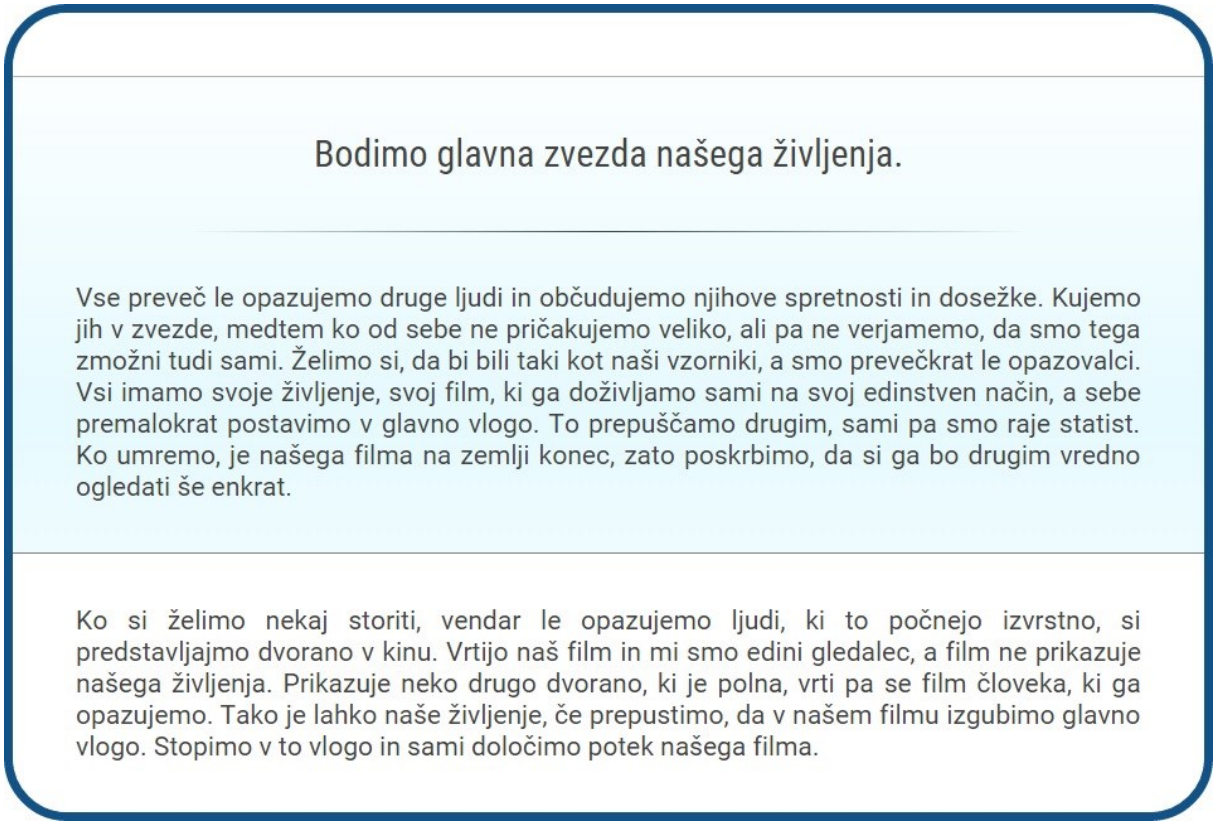
Slika 6.6.2: Če nas nekdo namenoma ponižuje, za to nismo krivi mi.



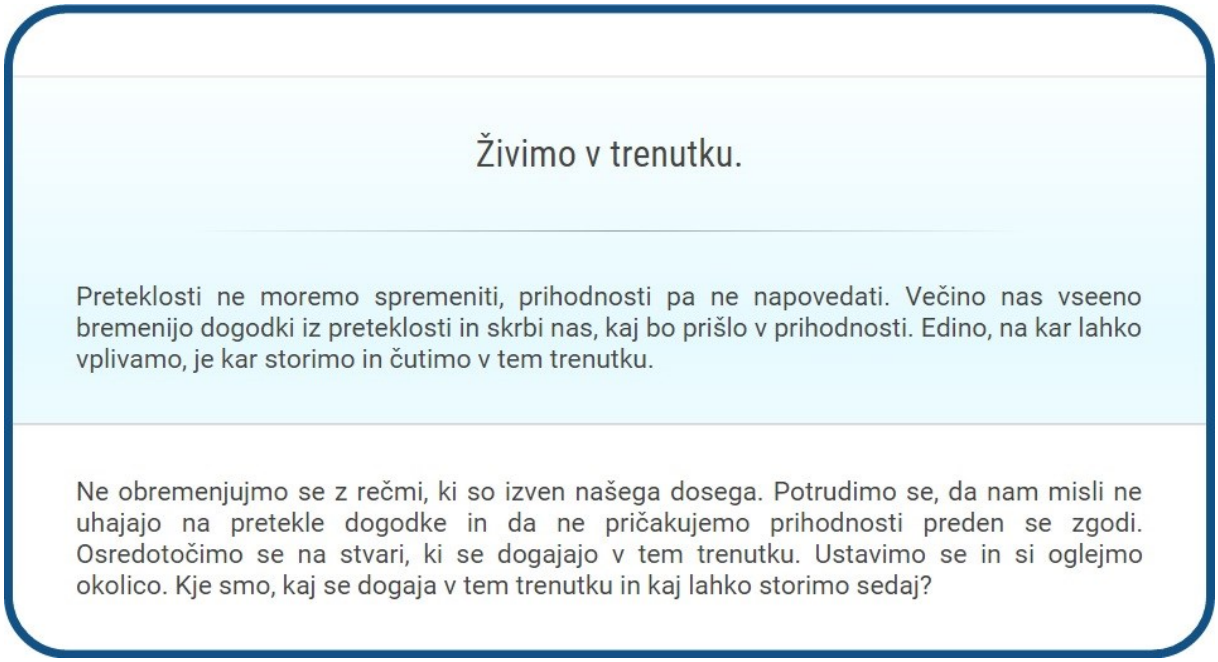
Slika 6.6.3: Misli oblikujejo naš svet.



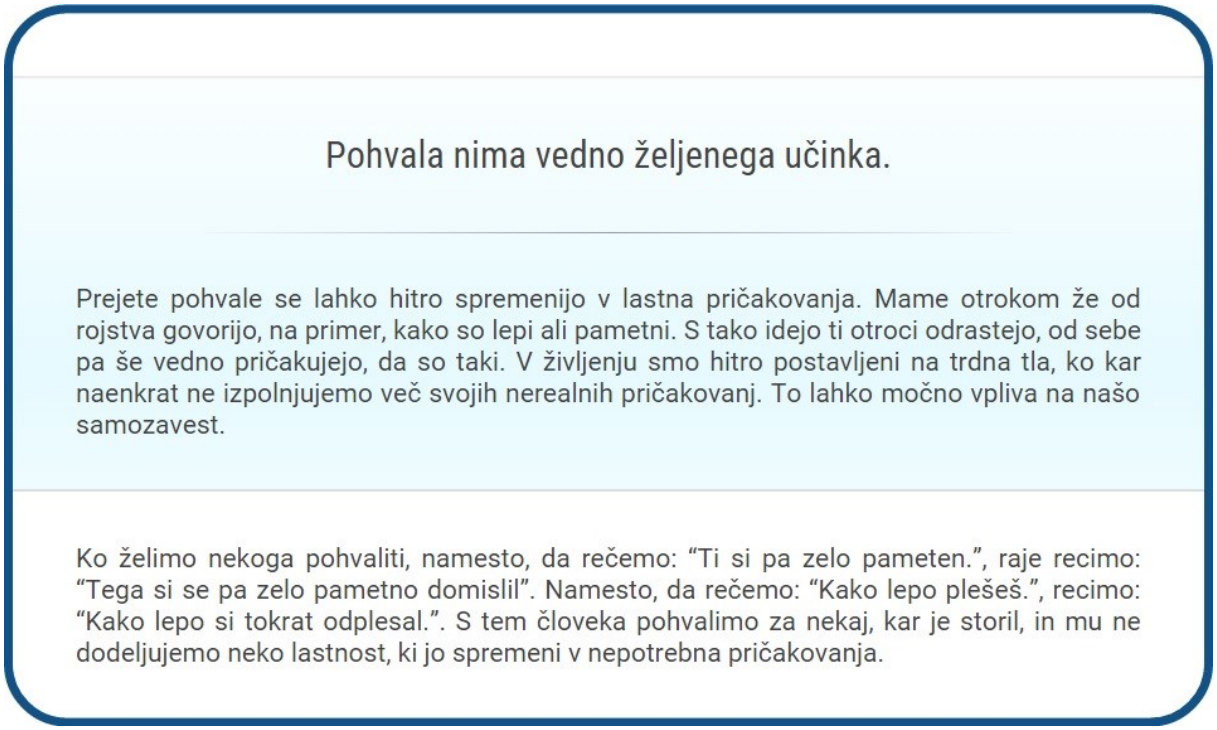
Slika 6.6.4: Zaupati moramo lastni presoji in na stvari gledati z lastnimi očmi[18].



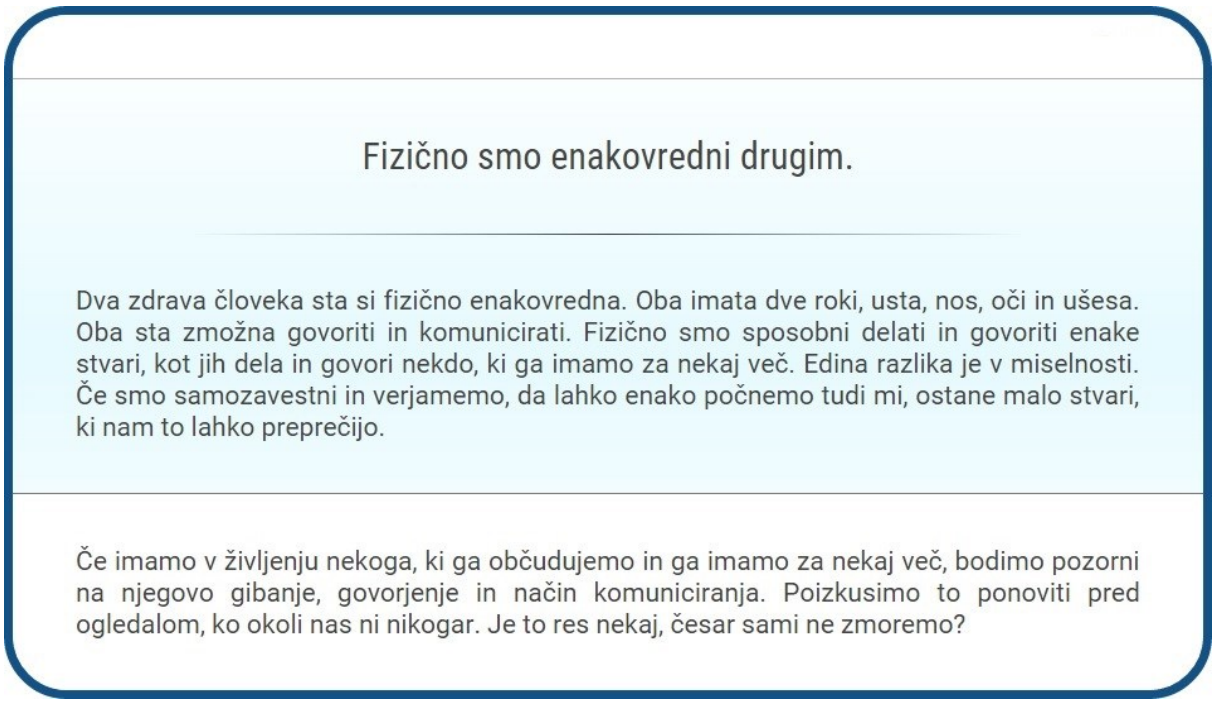
Slika 6.6.5: Bodimo glavna zvezda našega življenja.



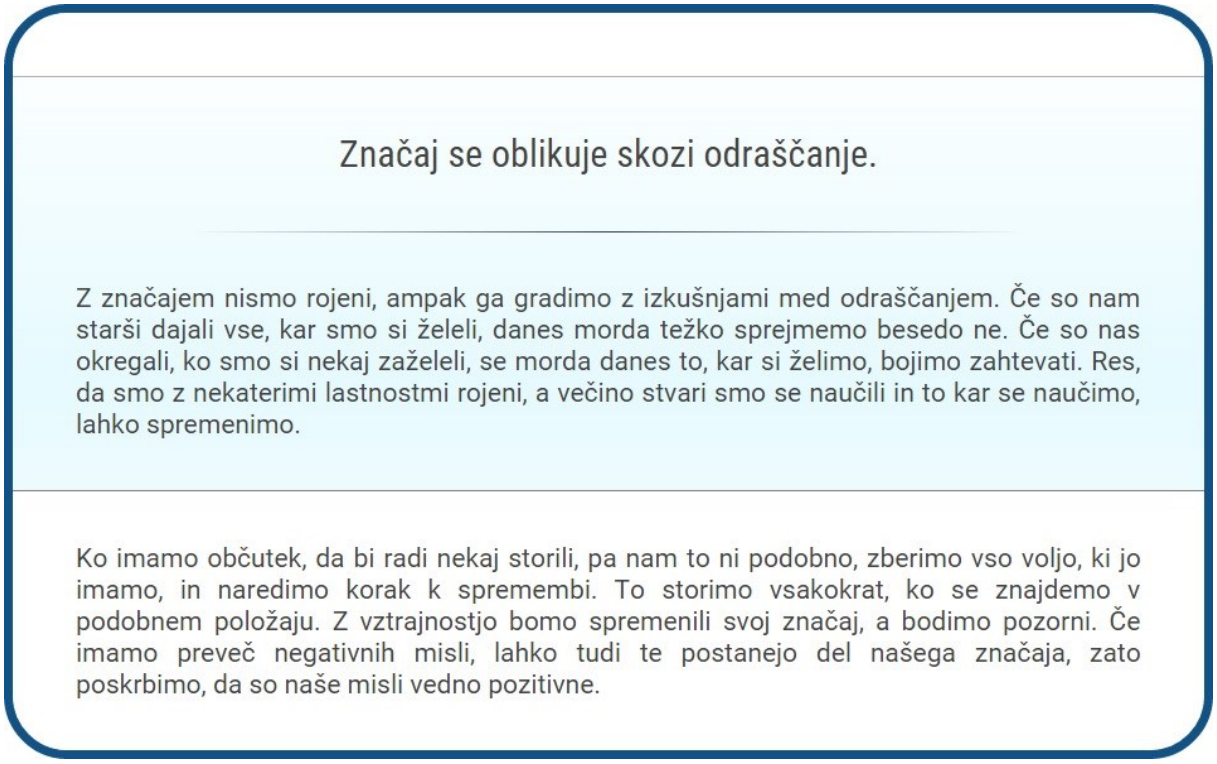
Slika 6.6.6: Živimo v trenutku.



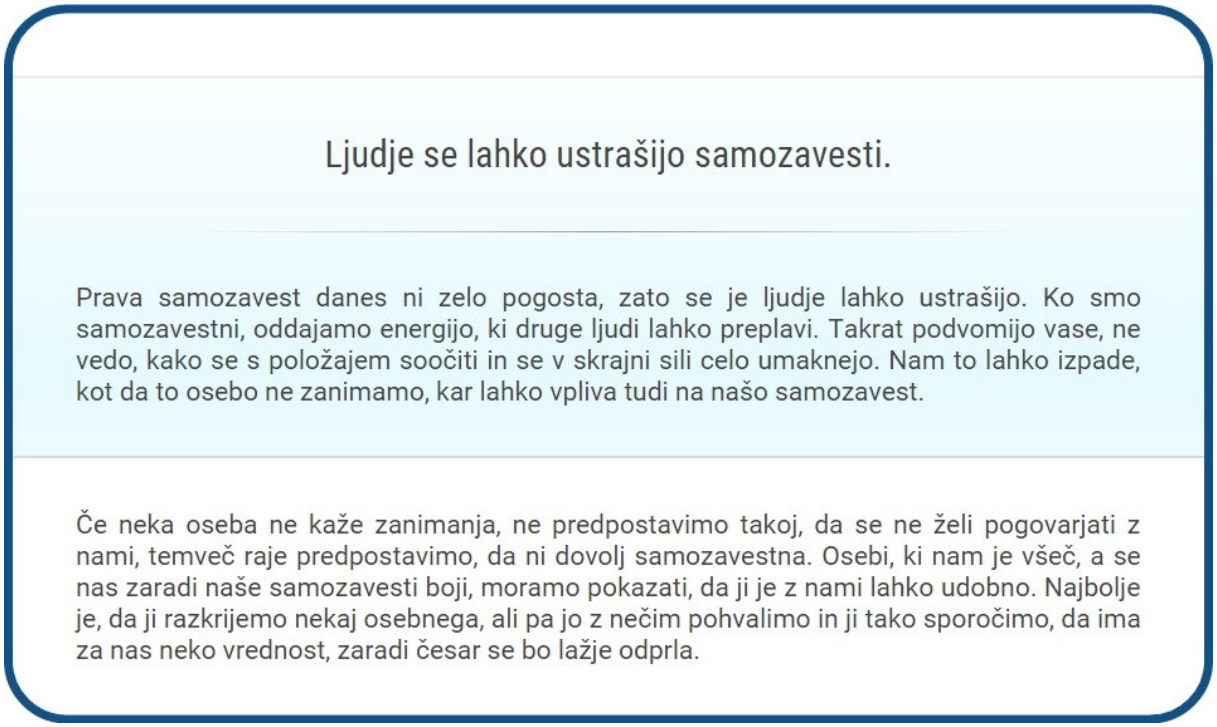
Slika 6.6.7: Pohvala nima vedno željenega učinka[1].



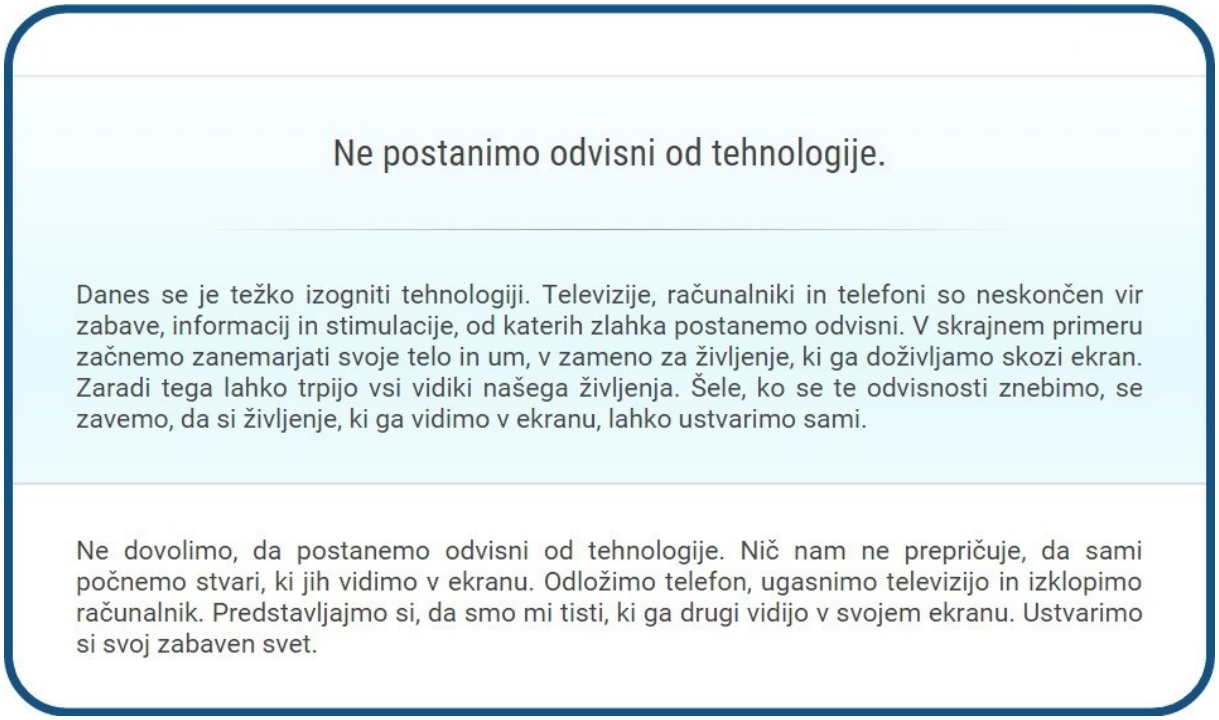
Slika 6.6.8: Fizično smo enakovredni drugim.



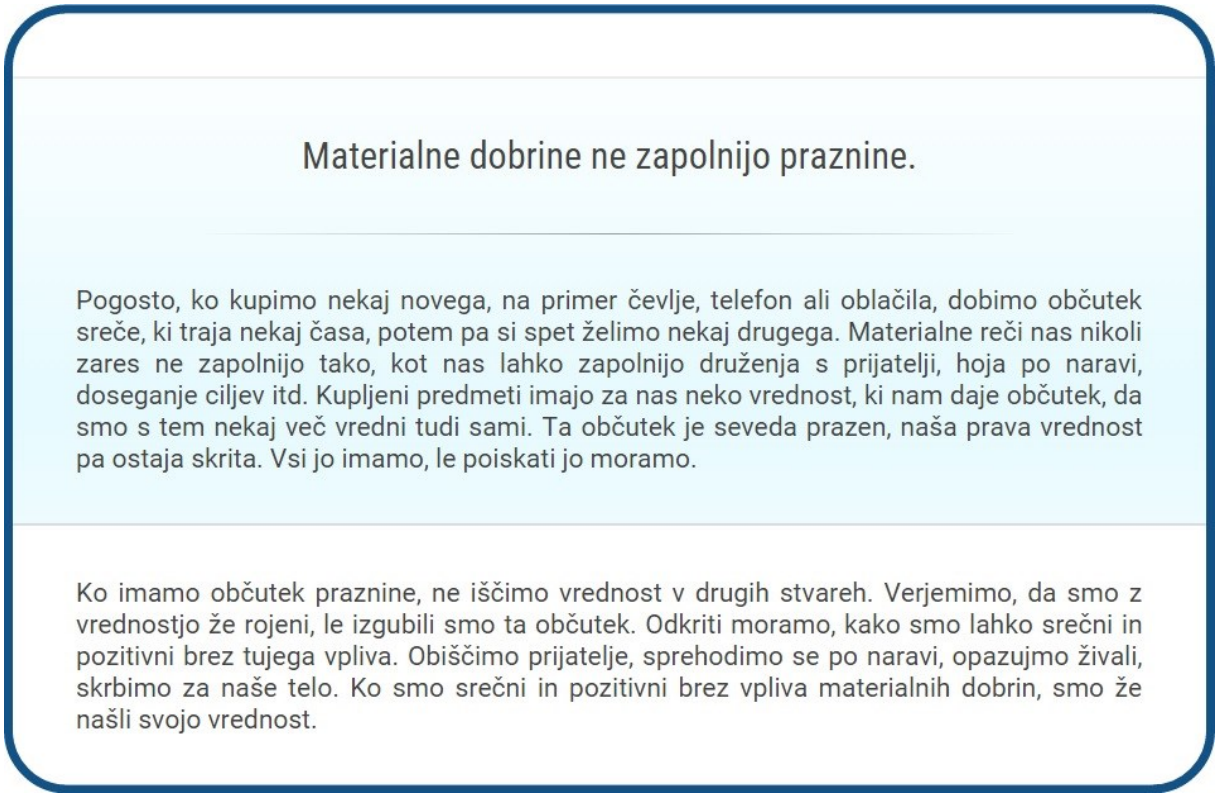
Slika 6.6.9: Značaj se oblikuje skozi odraščanje[1].



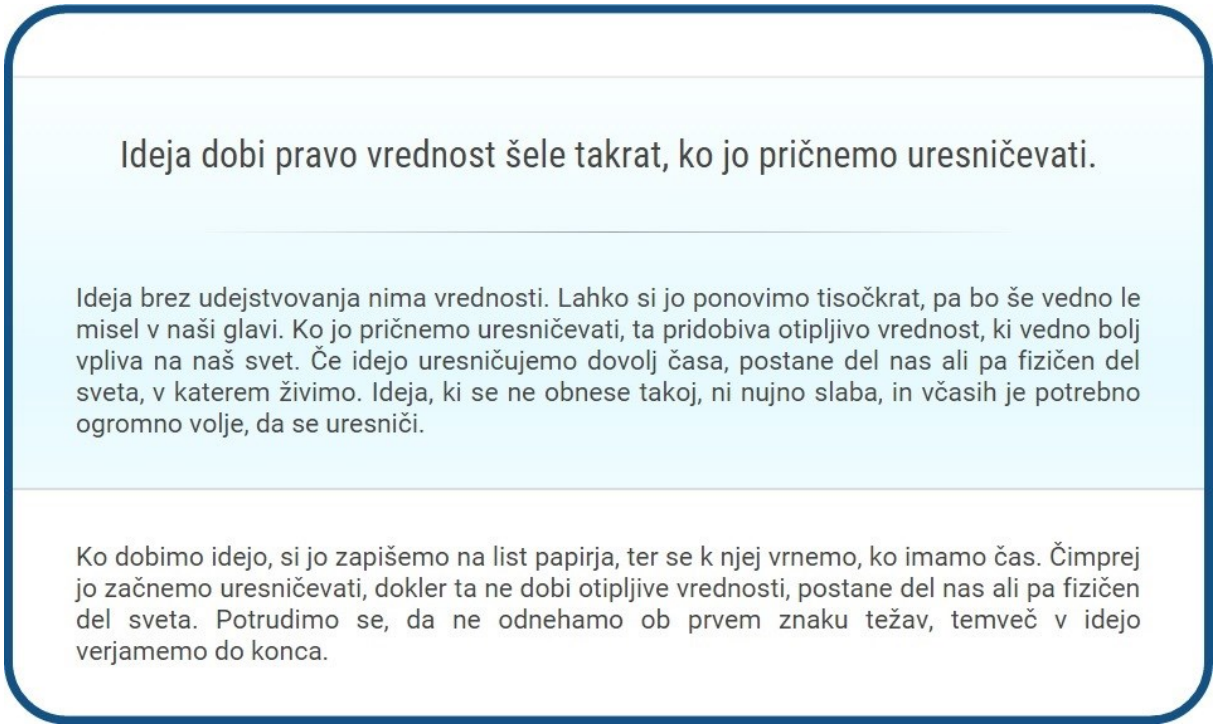
Slika 6.6.10: Ljudje se lahko ustrašijo samozavesti.



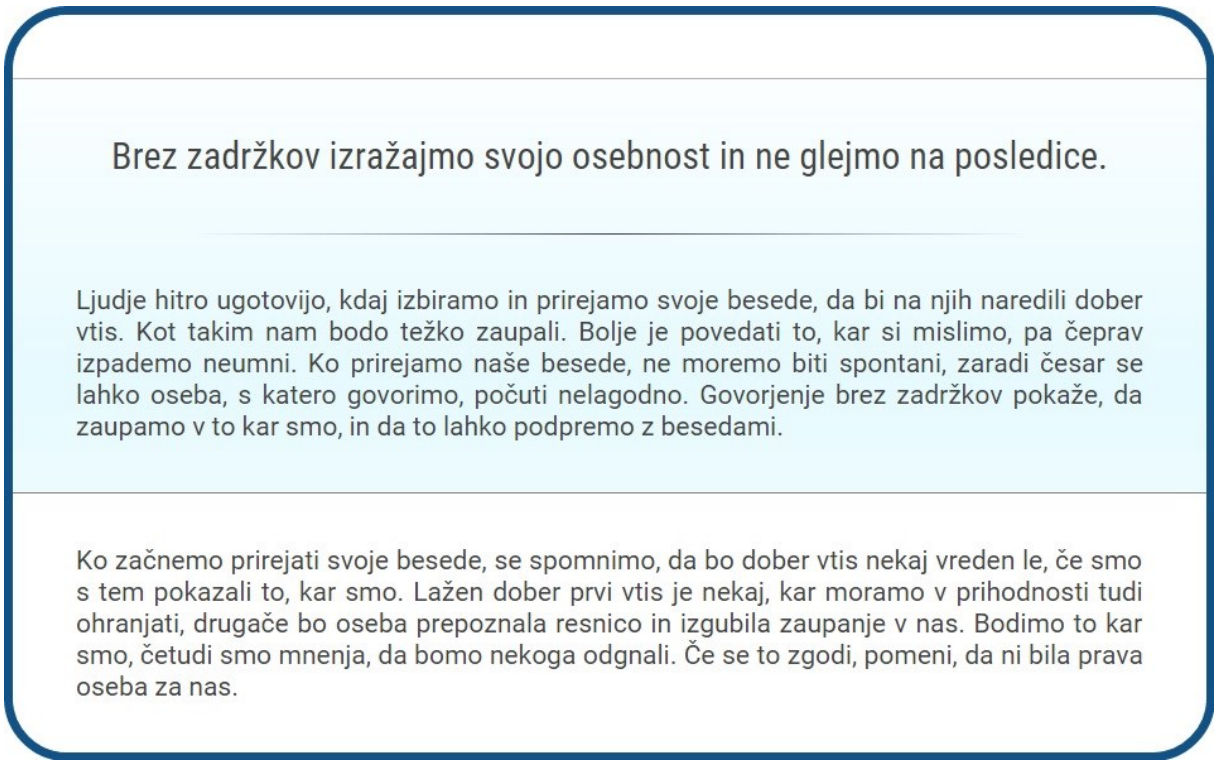
Slika 6.6.11: Ne postanimo odvisni od tehnologije.



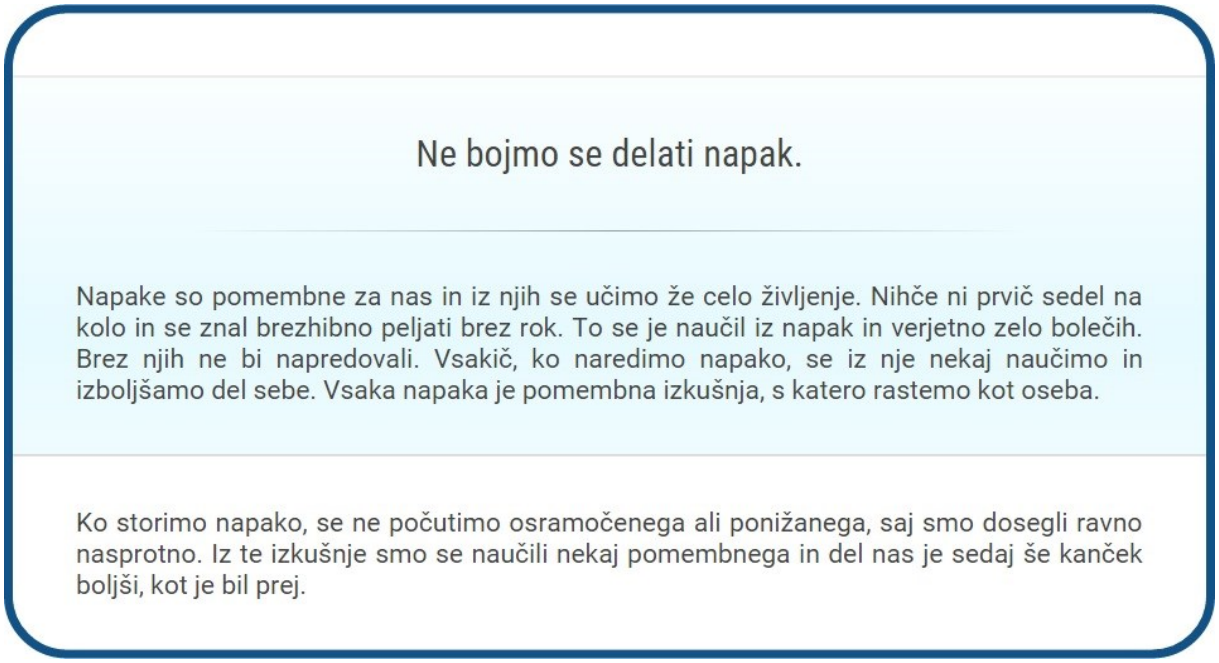
Slika 6.6.12: Materialne dobrine ne zapolnijo praznine[1].



Slika 6.6.13: Ideja dobi pravo vrednost šele takrat, ko jo pričnemo uresničevati[18].

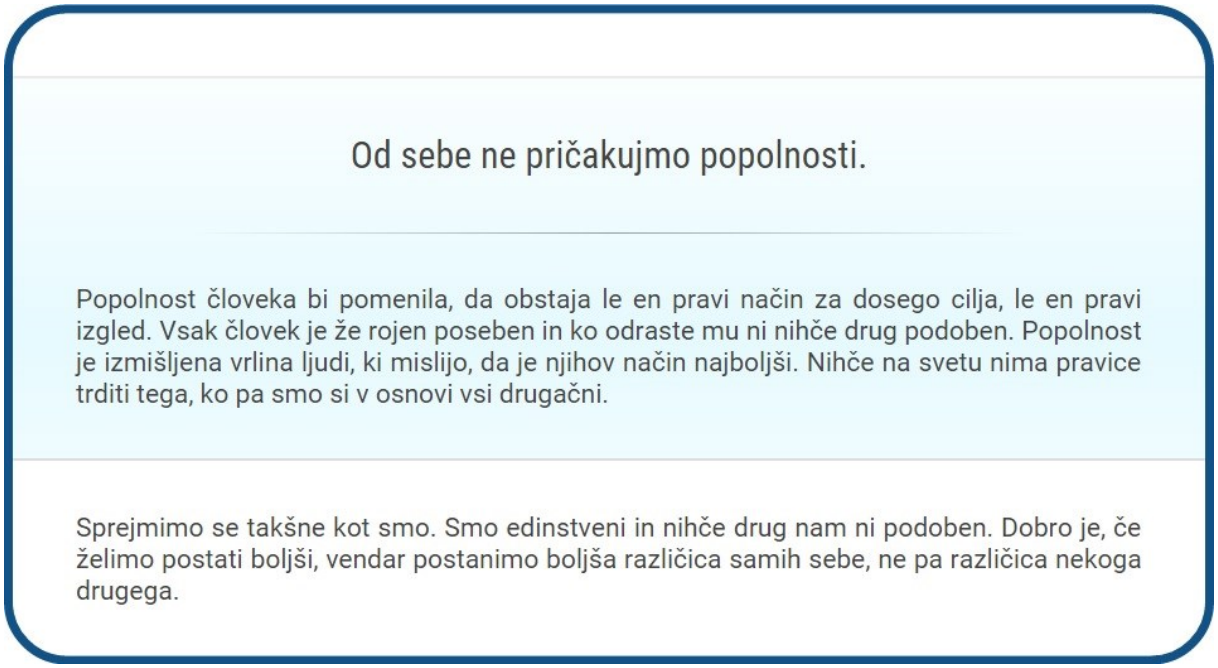


Slika 6.6.14: Brez zadržkov izražajmo svojo osebnost in ne glejmo na posledice[19].

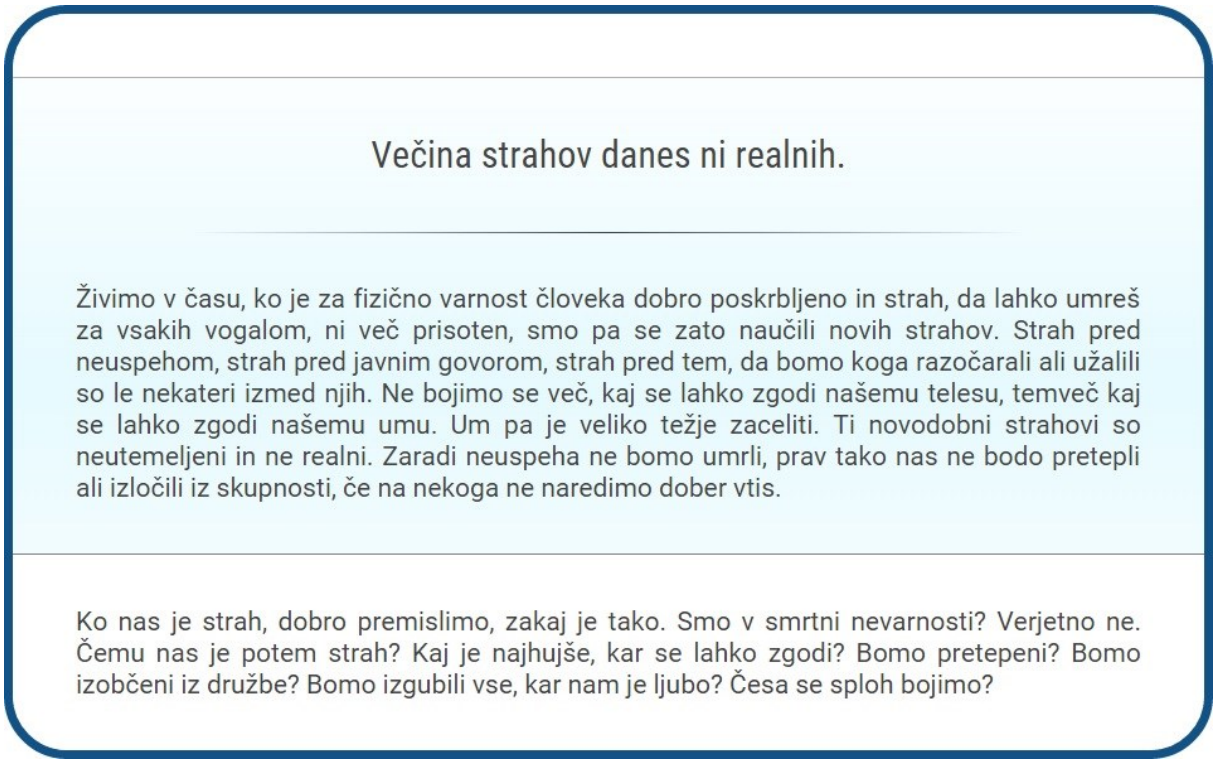


Slika 6.6.15: Ne bojmo se delati napak[1].

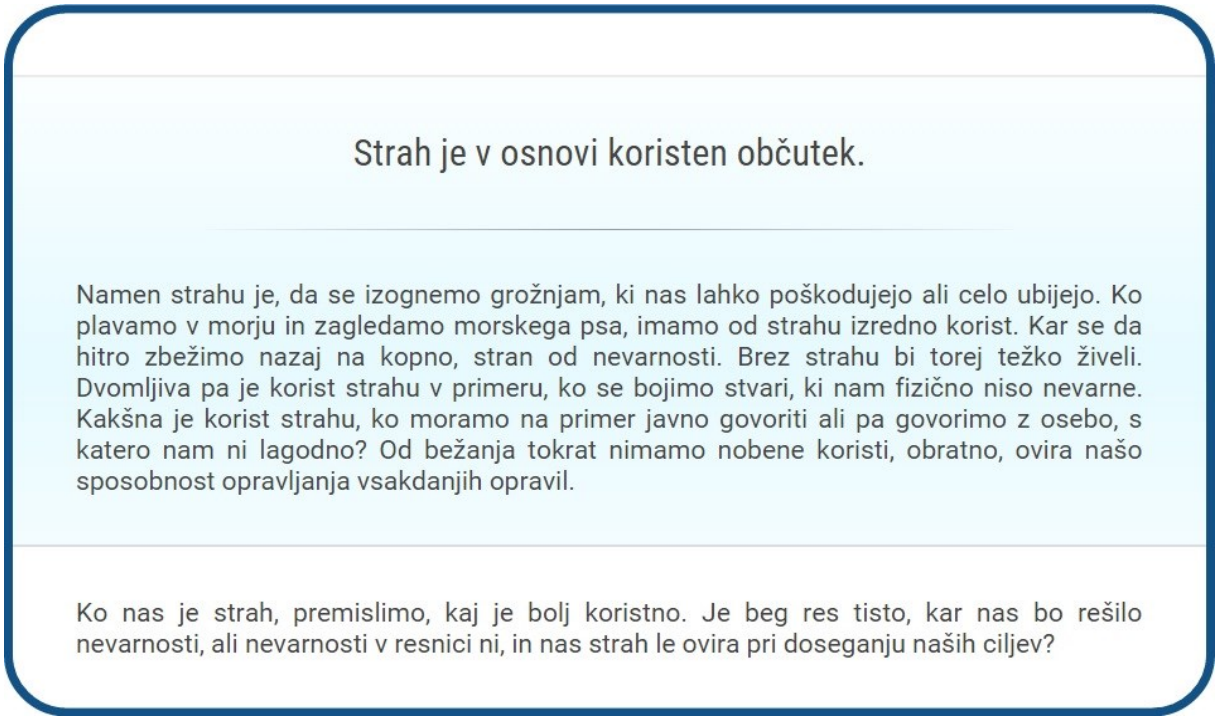
Slika 6.6.16: Sposobni moramo biti živeti sami[20].



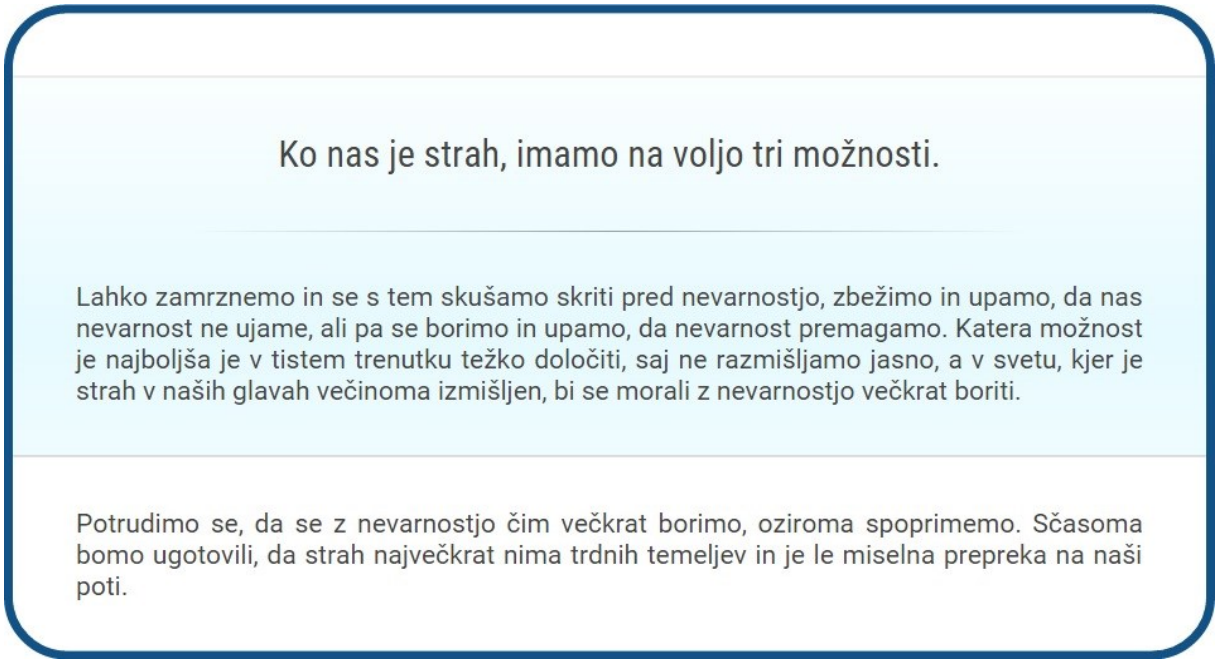
Slika 6.6.17: Od sebe ne pričakujmo popolnosti[1].



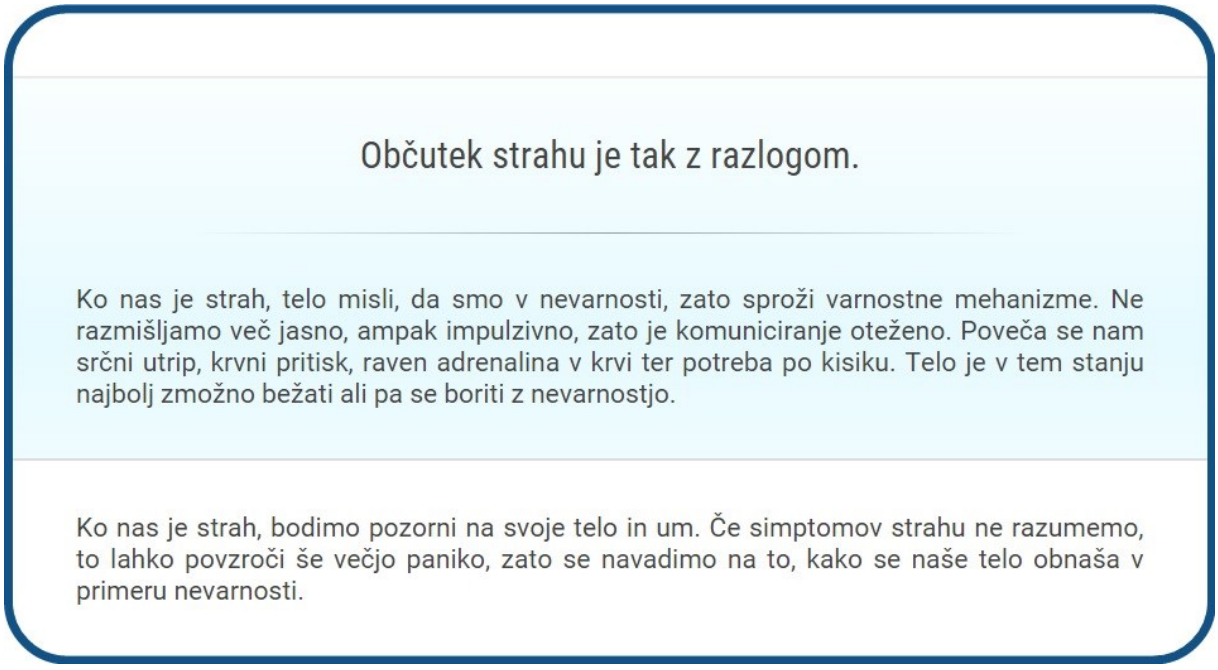
Slika 6.6.18: Večina strahov danes ni realnih[21].



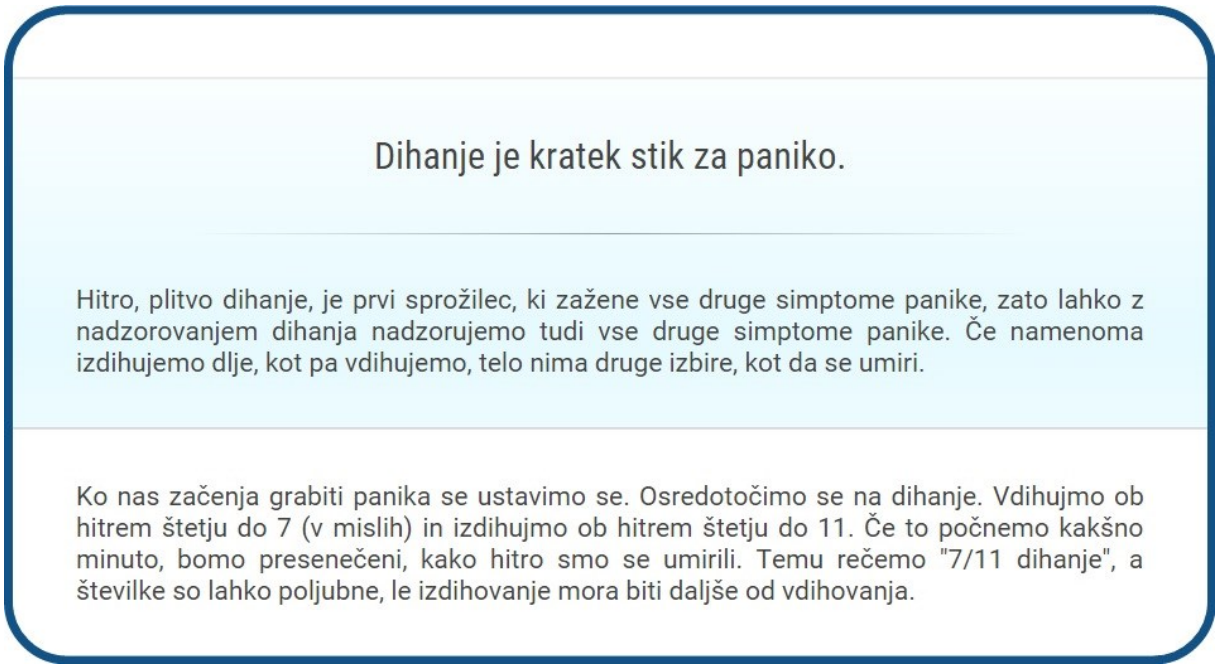
Slika 6.6.19: Strah je v osnovi koristen občutek[21].



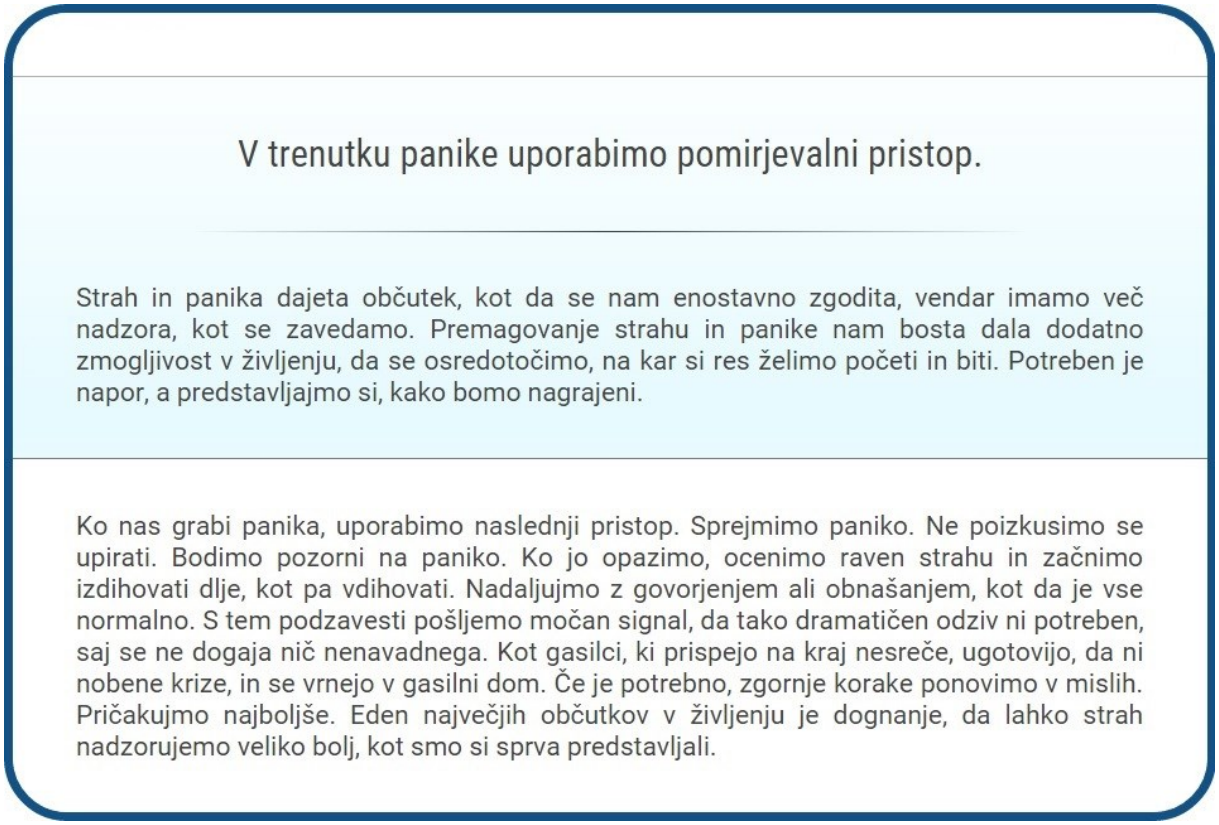
Slika 6.6.20: Ko nas je strah, imamo na voljo tri možnosti[21].



Slika 6.6.21: Občutek strahu je tak z razlogom[21].



Slika 6.6.22: Dihanje je kratek stik za paniko[22].



Slika 6.6.23: V trenutku panike uporabimo pomirjevalni pristop[22].

Poglavje 7 Sklepne ugotovitve

Mnenja smo, da ima aplikacija ogromen potencial, ki ga trenutno nismo zmožni doseči sami, zato je aplikacija le delček tistega, kar bi lahko bila. Razvili smo jo v prvi vrsti iz osebnih razlogov. Želeli smo ustvariti nekaj, kar bo imelo pozitiven vpliv na nas, našo družino in prijatelje. Končni produkt ni primeren le za nas, temveč za vsakega človeka, ki se zaveda, da so misli in občutki rezultat našega psihološkega stanja, ki ga s trudom in časom lahko spremenimo.

Aplikacija lahko veliko prispeva družbi. Če jo uporabljajo otroci in mladostniki, jim ni potrebno čakati več desetletij, da se naučijo istih naukov in idej, ki jih v življenju razkrijejo njihovi straši, ampak lahko nadgrajujejo in za naslednjo generacijo ustvarjajo čisto nova, sedaj še nepoznana psihološka razkritja. Tako mladi kot odrasli si lahko pomagajo v primeru krize identitete. Sploh danes je to velik problem, ko dajemo vedno večjo vrednost materialnim dobrinam, sebe pa zapostavljamo. S pomočjo aplikacije spoznajo sebe in svoja čustva ter ugotovijo, da materialnost ni pomembnejša od tega, kar nosijo v sebi. Poleg večanja vrednosti materialnih dobrin je pogosto težava tudi v preveliki odvisnosti med ljudmi. Ko se na nekoga preveč navežemo, lahko od njega postanemo odvisni, ko nas zapusti, pa imamo občutek, da smo izgubili del sebe. Tudi tu lahko aplikacija prispeva pomembna razkritja, ki človeku pomagajo, da v odnosih postavi trdne meje, ki se jih ne prestopa, če pa jih, se tega vsaj zaveda.

Najbolj osnoven namen aplikacije je uporabniku ponuditi odmerek pozitivnosti, kadarkoli ga potrebuje. Želimo, da se vrača vsak dan in da se po vsaki uporabi počuti bolje. Vsakič, ko doživi nekaj, česar ne razume, ali pa ne ve, kako se s tem spoprijeti, ima priložnost, da ne ostane ujet v svoji glavi, temveč odkrije nekaj novega, kar mu odpre svet. S časom bo razvil svojo čustveno inteligenco in s svojimi idejami bo začel vračati okolju, ki mu je ponudilo priložnost osebnostne rasti. Aplikacijo lahko uporabljajo otroci in mladoletniki, ki sveta še ne poznajo in si niti ne predstavljajo razsežnosti, ki jih ponuja njihov um. Prav tako jo lahko uporabljajo odrasli ljudje, ki imajo stresne službe in odnose, travme iz preteklosti ali karkoli drugega, kar jim otežuje življenje v smislu psihe. Nenazadnje lahko aplikacijo uporabljajo tudi ljudje, ki pomoči ne potrebujejo, so pa dovolj pametni, da se zavedajo pomena psihične stabilnosti in so morda sami pripravljeni pomagati ljudem v stiski.

Diplomsko delo povezuje računalništvo in psihologijo. Internet je v našem primeru medij za prenos idej, ki doseže veliko več ljudi, kot to lahko storijo fizični mediji, na primer knjige. To nam seveda pomaga, da se dotaknemo ogromno ljudi, a zraven prinaša druge težave. Kot smo omenili v eni od osnovnih rezin, se lahko ideje nenadzorovano širijo kot plamen, še toliko bolj danes, ko smo le klik stran od informacij. Ljudje imamo nevarno nagnjenje, da verjamemo skoraj vsemu, kar preberemo, če pa se teorija v praksi ne izide, pogosto dvomimo v svoje sposobnosti in ne v prebrane ideje. Ponujanje psihološke pomoči je pravzaprav poseg v intimno stran človeka, ki jo skrivamo. Veliko ljudi, ki bodo aplikacijo uporabljali, se bo za to odločilo, ker se morda počutijo šibke in ranljive ter si želijo najti vsebino, ki bi jim pomagala. To je seveda namen aplikacije, vendar se s psihologijo ranljivega uporabnika ni za igrati, sploh če je dovzeten za negativne ideje. Vsak uporabnik oziroma čutilec lahko piše svoje ideje, ki pa morda odražajo njegovo negativno plat. Če taka ideja pride pred oči ranljivega čutilca, lahko povzroči ogromno škodo, zato bi bilo najbolje, da vsako idejo pred objavo pregleda strokovnjak psihologije, ki je sposoben oceniti, kakšen vpliv ima ta lahko na posameznika. Razlog več za to je tudi ta, da psihologi v praksi ne uporabljajo takega pristopa, temveč obravnavano osebo najprej preučijo z uporabo različnih psiholoških testov, ki pokažejo njegovo psihično stanje. Šele po tem se odločijo, kaj je najboljši način, da se obravnavani osebi pomaga. V aplikaciji smo se temu izognili tako, da ponujamo ideje za različne vrste mišljenja, vendar to pomeni, da mora uporabnik sam vedeti, kaj potrebuje, ali pa to odkriva sproti. Sistem ima zato očitne pomanjkljivosti, na katere bomo morali biti pozorni pri nadaljnjem razvoju.

Projekta razvoja aplikacije smo se lotili brez konkretnega predhodnega znanja. Lahko bi izbrali tehnologije, ki so že dolgo časa v uporabi, a če smo se bili primorani učiti od začetka, naj bodo to tehnologije, ki definirajo nove generacije. Node.js in Express.js smo uporabili le kot vmesnik med podatkovno bazo in odjemalcem, zato imamo namen v prihodnosti uporabiti njuno polno moč. Možnost, da v celotni aplikaciji uporabljamo le en programski jezik, se nam zdi izredno plodna. Želimo si preizkusiti tudi tehnologije, kot sta Ruby on Rails ali pa Django, čeprav nista več tako popularni, sta nam pa oba jezika, tako Ruby kot Python, ki ju ti tehnologiji uporabljata, izredno privlačna.

Pri podatkovni bazi smo uporabili ogromno denormalizacije, kar pomeni, da dokumentov v zbirkah nismo referencirali s ključem, kot se to počne pri mehanizmu SQL, temveč smo informacije dokumentov vedno znova ponavljali. Primer je rezina, ki jo doda vsak čutilec. Dokument te rezine je sedaj v zbirki dodanih rezin vseh čutilcev, kar se sliši zelo redundantno, a protiutež je ta, da nam ni potrebno vsake rezine posebej potegniti iz podatkovne baze. Rezultat tega naj bi bilo zelo hitro poizvedovanje, vendar obstaja možnost, da v našem primeru ni tako, saj je shema podatkovne baze, ki smo jo zasnovali, daleč od

popolne, in bi morda poizvedovanje delovalo še hitreje, če bi uporabili mehanizem SQL, s katerim smo imeli že izkušnje. Ideja, da sheme ni potrebno definirati, se lahko začetniku zdi dobra, vendar se slej kot prej prikažejo posledice premalo podrobnega načrtovanja. V končani shemi podatkovne baze je opaziti pomanjkanje vidika varnosti, poleg nastavljanja edinstvenih vrednosti in njihovih tipov. Varnosti v mehanizmu NoSQL nismo dovolj raziskali in je absolutno eden glavnih področij, ki se jim moramo še posvetiti.

Najbolj zadovoljni smo z uspehom uporabe ogrodja AngularJS. Učenju delovanja le-tega smo posvetili res ogromno časa in večkrat se je pojavil dvom o njegovi uporabi. Obstaja več veliko bolj enostavnih načinov izdelave odjemalnega dela, v skrajnem primeru bi lahko uporabili sintakso HTML brez posebne nadgradnje, vendar smo si zares želeli stopiti v korak s časom. AngularJS nam je omogočil zelo hitro realizacijo zamišljenih funkcionalnosti, izkazal pa se je za odlično ogrodje pri razvoju visoko interaktivne aplikacije. Kljub vsemu je treba poudariti, da bi lahko izbrali še boljšo pot. Obstajajo tehnologije, ki se razvijanja odjemalnega dela lotevajo na drugačen način, kar nas sicer ne bi zanimalo, če ne bi bila v razvoju tudi naslednja različica ogrodja AngularJS, ki uporablja prav te novejše principe. Podpora za prvo različico, kot kaže, že počasi upada, kar pomeni, da bomo morali slej kot prej nadgraditi znanje, v najslabšem primeru pa je bilo učenje prve različice AngularJS zaman. Vseeno bomo obdržali občutek, da smo se zmožni naučiti katerega koli ogrodja, ter ga uspešno uporabiti v praksi.

7.1 Odzivi testnih uporabnikov

Preden smo aplikacijo testirali z uporabniki, smo nekaj ljudi povprašali o ustreznosti naših osnovnih rezin. Glede na to, da smo relativno mladi in nismo študirali psihologije, je bila večina nad vsebino pozitivno presenečena. Čudili so se, kako da razumemo stvari, ki so jih oni spoznali šele pri petdesetih letih, a to je seveda posledica tega, da se danes zelo malo poudarka daje na psihološko stanje ljudi. K sreči je odnos do tega iz leta v leto boljši.

Aplikacijo smo izpostavili na socialnih omrežjih in forumih. Obiskalo jo je kar nekaj ljudi, a smo dobili občutek, da je bila večina na splošno malo zadržana. Čustva in občutki so nekaj zelo osebnega in ni se vsakemu lahko soočiti z njimi, zato mora biti zelo očitno, da je namen aplikacije izključno pozitiven in da je uporabnik med uporabo popolnoma varen. Ocenjujemo, da se je le petina registriranih uporabnikov vrnila po prvem obisku, kar je za nas razumljivo. Vsi so spremenili misel in dodali vsaj eno rezino, a nekaterih osebnostna rast in razumevanje čustev enostavno ne zanimata, nekateri na to niso pripravljeni, a se bodo nekoč le vrnili, obstajajo pa taki, ki so v konceptu aplikacije takoj videli vrednost in so tudi napisali kakšno svojo rezino. Taki so nam povedali tudi, da se jim aplikacija zdi izredno motivacijska.

Ves čas smo imeli pogled na podatkovno bazo, tako da smo videli, kaj počnejo uporabniki. To se nam sicer sliši kot nespodobno dejanje, a ker smo vpisovanje osebnih informacij primerno omejili, nismo nikoli vedeli, kdo so uporabniki v resnici. Skrb glede tega so nam izrazili nekateri uporabniki, ki so mnenja, da je na internetu že tako slabo poskrbljeno za varnost podatkov, sedaj pa želimo iz njih pridobiti še najbolj osebne podatke. Strinjali smo se, da je skrb primerna in v prihodnosti je potrebno glede tega dobro premisliti. Literatura[20] sicer pravi, da je premalo ljudi pripravljenih pokazati svoja čustva, kar lahko slabo vpliva na njih in ljudi okoli njih. Iz podatkov smo videli, da so si uporabniki takoj spremenili misel, s čimer smo bili izredno zadovoljni. Za nas to pomeni, da uporabnik takoj vidi, da je stvar interaktivna in da si jo lahko priredi po svojih željah.

Dobili smo veliko odzivov na barvo in obliko aplikacije, ki se je vsem zdela dobra in primerna. Dobili smo tudi odzive na funkcionalnost aplikacije in veliko mnenj smo takoj upoštevali. Širina pogledov je bila preširoka in med elementi je bilo preveč prostora, zato smo širino zmanjšali. Uporabniki so bili zbegani nad praznim prikazom zadnjih aktivnosti, zato se ta element sedaj pokaže šele takrat, ko kakšne aktivnosti dejansko obstajajo. Koncept dosežkov jim ni bil dovolj razumljiv, zato bo potrebno premisliti, kako za njih ustvariti večji pomen in čemu dejansko služijo slike v profilki. Sama struktura rezin jim je bila razumljiva in so si jih tudi veliko dodali, nekateri pa so celo ustvarili zelo dobre rezine, ki so takoj vplivale tudi na naše mišljenje. To je dokaz, da stvar deluje, in prepričani smo, da imajo naše osnovne rezine enak vpliv na uporabnike, kot imajo njihove rezine na nas.

7.2 Cilji za nadaljnji razvoj

Iz mnenj uporabnikov smo razvili dobro podlago za nadaljnje delo. Aplikacijo bi bilo dobro razviti še v angleščini in jo tako ponuditi tudi izven okvirjev Slovenije. Dobro bi bilo, da se razvije nekakšna klepetalnica ali forum, da se ima uporabnik možnost z nekom pogovarjati. Tako se ne bi vračal le zaradi vsebine, temveč tudi zaradi prijateljev. Na voljo bi lahko bila spletna psihološka svetovalnica, kjer strokovnjak odgovarja na vprašanja uporabnikov, ki imajo specifične težave. Mogoče bi celo dopustili možnost, da na vprašanja odgovarjajo drugi uporabniki, ki imajo ogromno dosežkov in so se v skupnosti znotraj aplikacije izredno izkazali.

Z aplikacijo smo zadovoljni, ker se zavedamo, koliko časa smo vanjo vložili, odzivi so bili pozitivni, pot pa je še dolga, saj ima ogromna neizkoriščenega potenciala. Namesto podatkovne baze MongoDB si želimo v aplikaciji preizkusiti MySQL. V preteklosti smo imeli z njo veliko uspeha in zanima nas, kako bi se obnesla pri takem tipu aplikacije, kjer se

hitro izmenjuje veliko podatkov. MongoDB nam je tu dobro služila, zato želimo nekoč nadgraditi tudi znanje podatkovnih baz, ki uporabljajo mehanizem NoSQL.

Obstaja veliko tehnologij, ki omogočajo sprotno testiranje razvite kode, a se nam je ta koncept zdel prevelik zalogaj, ki bi le upočasnil napredek. To razmišljanje verjetno ni bilo tako napačno, a danes zares vidimo vrednost sprotnega testiranja. Velikokrat je aplikacija brez razloga nehala delovati ali pa implementacija nove funkcionalnosti ni bila uspešna. Porabili smo tudi po več ur naenkrat, da smo na koncu odkrili, da v eni vrstici kode pri spremenljivki manjka črka. Testiranje je eden prvih konceptov, ki jih bomo v nadaljevanju implementirali.

Z vidika razvoja nas najbolj moti dejstvo, da velikost aplikacije v brskalniku ni fleksibilna na različnih napravah. Narejena je za računalniški ekran in čeprav brskalniki danes že sami ponujajo malo fleksibilnosti, to ni isto, kot če bi bila aplikacija zasnovana za vse vrste ekranov. Vsekakor je tudi to koncept, ki ga želimo čim prej implementirati.

Na strani odjemalca imamo namen preizkusiti nova ogrodja, ki uporabljajo nove principe in pridobivajo vedno večjo podporo. Naučili smo se uporabljati ogrodje AngularJS, sedaj pa mu že grozi zastarelost. Tega ne obžalujemo, nasprotno, navdihuje nas ideja, da se svet računalništva burno razvija, in z veseljem se bomo naučili uporabljati ogrodje AngularJS 2 ali katerega drugega.

V korist uporabnikov si želimo, da bi vse rezine pregledal usposobljeni strokovnjak psihologije. Tako bi bili prepričani, da te uporabnikom ne škodijo. En način, ki ga lahko hitro uvedemo, je ta, da bi morali vsako ustvarjeno rezino najprej potrditi mi, preden bi ta bila vidna drugim uporabnikom, pozneje pa bi dobili nekoga, ki je za tako odločanje dejansko kvalificiran.

Zaradi ogromne količine dela aplikacija ne vsebuje zelo pomembne funkcionalnosti odstranjevanja in spreminjanja rezin. Brez tega v praksi ne gre, čeprav se nad tem zaenkrat ni pritoževal noben od uporabnikov. Imeli smo ogromne težave pri razvijanju, zato smo se odločili, da to funkcionalnost izpustimo. Do težav je prišlo, ker nismo sproti testirali kode in nismo točno razumeli, kakšne so pravice in zahteve pri modulih za avtentikacijo uporabnika, tako da je to še eden od konceptov, ki jih moramo bolj podrobno raziskati.

Literatura

[1] Ljubica Uvodić-Vranić, *Pustolovščina osebne preobrazbe*, Ebesede, str. 11-28, 34-47, 82-86, 118-137 in 157-160, 2014.

[2] Node.js. [Online]. Dosegljivo: https://nodejs.org/en/.

[3] Chrome V8. [Online]. Dosegljivo: https://developers.google.com/v8/.

[4] Uvod v Node.js. [Online]. Dosegljivo: http://www.tutorialspoint.com/nodejs/nodejs_introduction.htm.

[5] Upravljalec paketov Node. [Online]. Dosegljivo: https://www.npmjs.com/.

[6] Express.js. [Online]. Dosegljivo: http://expressjs.com/.

[7] MongoDB. [Online]. Dosegljivo: https://www.mongodb.org/.

[8] Definicija MongoDB. [Online]. Dosegljivo: http://searchdatamanagement.techtarget.com/definition/MongoDB.

[9] Mongoose.js. [Online]. Dosegljivo: http://mongoosejs.com/.

[10] Passport. [Online]. Dosegljivo: http://passportjs.org/.

[11] JavaScript Web Token. [Online]. Dosegljivo: https://jwt.io/.

[12] AngularJS. [Online]. Dosegljivo: https://docs.angularjs.org/guide/introduction.

[13] JavaScript. [Online]. Dosegljivo: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction.

[14] jQuery. [Online]. Dosegljivo: http://www.tutorialspoint.com/jquery/jquery-overview.htm.

[15] HTML. [Online]. Dosegljivo: http://www.computerhope.com/jargon/h/html.htm.

[16] CSS. [Online]. Dosegljivo: http://html.net/tutorials/css/lesson1.php.

[17] Git. [Online]. Dosegljivo: http://git-scm.com/video/what-is-git.

[18] Napoleon Hill, *Think & Grow Rich*, Ballatine Books, str. 1-14 in 177-184, 1983.

[19] David Deida, *The Way of the Superior Man,* Sounds True, str. 14-50, 2004.

[20] David Wygant, *Naked!*, Hay House, str. 15-36, 2012.

[21] Understanding & Overcoming Fear. [Online]. Dosegljivo: http://www.eruptingmind.com/understanding-overcoming-fear/.

[22] 5 Sure-fire Ways to Overcome Fear and Anxiety Today. [Online]. Dosegljivo: http://www.uncommonhelp.me/articles/overcome-fear-and-anxiety/